0993-8155

PERKIN-ELMER ROBOT LANGUAGE (PERL) INSTRUCTIONS

August 1986

PERKIN-ELMER NORWALK, CONNECTICUT, U.S.A.

PERKIN-ELMER

CUSTOMER LICENSE AGREEMENT

THE PROGRAM FURNISHED HEREWITH IS LICENSED BY PERKIN-ELMER TO CUSTOMERS FOR THEIR USE ONLY ON THE TERMS AND CONDITIONS SET FORTH BELOW. OPENING THE DISKETTE PACKAGE INDICATES YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS.

1.0 DEFINITIONS

1.1 "Licensed Program" shall mean any Object Code supplied by LICENSOR under this License.

1.2 "Designated Equipment" shall mean a computer system manufactured by PERKIN-ELMER.

1.3 "Use" shall mean the copying of any portion of Licensed Program from a storage unit or media into the Designated Equipment and executing any portion of Licensed Program on the Designated Equipment.

2.0 LICENSE GRANT

2.1 LICENSEE is granted a non-transferable, non-exclusive right to Use the Licensed Program supplied by LICENSOR on the Designated Equipment. LICENSEE may make copies of the Licensed Program as necessary for archive, testing, or backup purposes only.

3.0 TERM

3.1 This LICENSE shall be effective from the date of receipt of the LICENSED PROGRAM until it is terminated in accordance with Article 7.0.

4.0 PROPRIETARY RIGHTS AND PROTECTION

4.1 LICENSEE acknowledges and understands that nothing contained in this License shall be construed as conveying title in Licensed Program to the LICENSEE.

4.2 LICENSEE agrees to hold the Licensed Program in confidence and shall not disclose the Licensed Program, or part thereof, without the prior written approval of LICENSOR, except to its employees to whom disclosure is necessary for Use. Licensee agrees not to disassemble, decompile, or cross-compile Licensed Program.

4.3 Where any of LICENSOR'S rights to the Licensed Program, or part thereof, arise under an agreement with a third party supplier, such supplier shall have the benefit of LICENSOR'S rights hereunder.

4.4 LICENSEE acknowledges that the Licensed Program may contain certain encryptions, or other devices which may prevent or detect unauthorized Use of the Licensed Program. Temporary Use on backup equipment may require assistance from Licensor.

5.0 NOTICES AND LEGENDS

5.1 LICENSEE agrees to reproduce and include any copyright notices, trademarks, or other legends or credits included in the Licensed Program, or requested in writing by LICENSOR, in and on every copy made by LICENSEE of Licensed Program, or part thereof. The existence of any copyright notice shall not be construed as an admission or presumption of publication of the Licensed Program.

6.0 WARRANTY

6.1 LICENSOR warrants that for ninety (90) days from the date the Licensed Program is received, the Licensed Program supplied by LICENSOR shall perform in accordance with the standards set forth in the user manual related to the Licensed Program, if any, provided that the attached Software Registration Form has been returned to LICENSOR.

6.2 The above warranty does not extend and shall not apply to Licensed Program to the extent that any breach of warranty is caused by the Licensed Program being: (a) not Used in accordance with the user manual supplied hereunder; (b) Used in combination with any program material not licensed herein; (c) modified by LICENSEE; or (d) Used with equipment other than the Designated Equipment.

6.3 LICENSOR MAKES NO WARRANTIES THAT ERRORS HAVE BEEN COMPLETELY ELIMINATED FROM THE LICENSED PROGRAM OR THAT USE OF ANY LICENSED PROGRAM WILL NOT INFRINGE UPON ANY PATENT, COPYRIGHT, OR TRADEMARK. LICENSOR MAKES NO OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO FITNESS FOR A PARTICULAR PURPOSE OR MERCHANTABILITY WITH RESPECT TO THE LICENSED PROGRAM. THIS LIMITED WARRANTY GIVES LICENSEE SPECIFIC LEGAL RIGHTS, AND LICENSEE MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE

7.0 TERMINATION

7.1 If LICENSEE fails to comply with any term or condition of this License, LICENSOR may notify LICENSEE in writing of such failure. If LICENSEE fails to cure such failure within thirty (30) days of such written notice, this License may be terminated.

7.2 LICENSEE may terminate this License at will on thirty (30) days prior written notice to LICENSOR. Within ten (10) days after termination, LICENSEE shall certify in writing to LICENSOR that all copies, in whole or in part, in any form, of the Licensed Program have been destroyed or returned to LICENSOR.

CONTINUED ON OTHER SIDE

SOFTWARE CHANGE REQUEST

General Information

Name ________________________________ PERL version no. ______

Address ____

Phone ____________________ Check one: bug ____ enhancement ____

System Configuration

Circle one: IBM PC IBM XT IBM AT Clone ____________________

System memory size: ____________ K

Memory allocation:

stack	____________	K
program	____________	K
directory	____________	K

Communication ports installed (circle): 1 2 3 4 5 6 7 8 9 10

Problem or Suggestion

Error message received (if any):

Description of problem or suggested improvement:

Documentation included (please check)

Listings (required): Program ____ PERL.SCF ____

Disk copy (required for reporting a directory problem; otherwise, optional):

_____ PERL.DIR

_____ PERL.SCF

_____ Procedures (Please List)

Mail to:

Laboratory Robotics Division
The Perkin-Elmer Corporation
761 Main Avenue
Norwalk, CT 06859-0927

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

ILLUSTRATIONS

ILLUSTRATIONS

TABLES

1 INTRODUCTION

1 INTRODUCTION

DESCRIPTION OF PERL

Characteristics

PERL (Perkin-Elmer Robot Language) is a new software language devised for use in the Model 9000 MasterLab Automated Sample Preparation System. It is specifically intended to facilitate communication in a robotics system.

The MasterLab System consists of a number of modules. Some of the modules in the system, such as the robot, the syringe station, the device interface, and the balance, are able to communicate via RS232 ports with the System Controller (IBM PC), as shown in Figure 1-1. A unique facility, the Teach Program, is available in PERL for each of these modules. The Teach Programs allow you to create new commands, such as "go_to_mixer", in the laboratory, and store them for use in applications programs (called "procedures"). The robot learns, through a simple interactive procedure, the location specified by "go_to_mixer". It goes to that location whenever it encounters the command in a PERL procedure. In the same way, the syringe, the balance, and other devices can be taught the commands necessary in order for them to accomplish their tasks.

Other modules (the mixer and the crimping station, for example) are shown in Figure 1-1 connected to the device interface. The device interface can monitor logic inputs from the crimping station and can turn the mixer or the crimping jaws on and off. Thus it is the device interface Teach Program which is used to create the commands which integrate such modules into the system.

Finally, there are modules which are simply locations to which the robot hand must reach. These are shown at the bottom of the diagram in

Figure 1-1. Racks, dispensing probes, and also the mixer and crimping station are examples of such locations. The robot Teach Program is used to teach the robot the necessary interactions with these modules.

It is the PERL Teach Programs which are in fact responsible for the flexibility of the MasterLab System. You can easily add modules to the system to meet your needs, because you can create commands which will integrate these modules into the system. In addition, you can program new applications for your existing system, creating new commands as needed.

The Automated Sample Preparation System can carry out many different sample preparation procedures in your laboratory. The programming language provided for your use is simple to use, so that you can program the procedures as readily as you devise them. For this reason, PERL appears similar to BASIC, a language which is easy to learn and with which you may already be familiar. Not only are many of the statements the same as

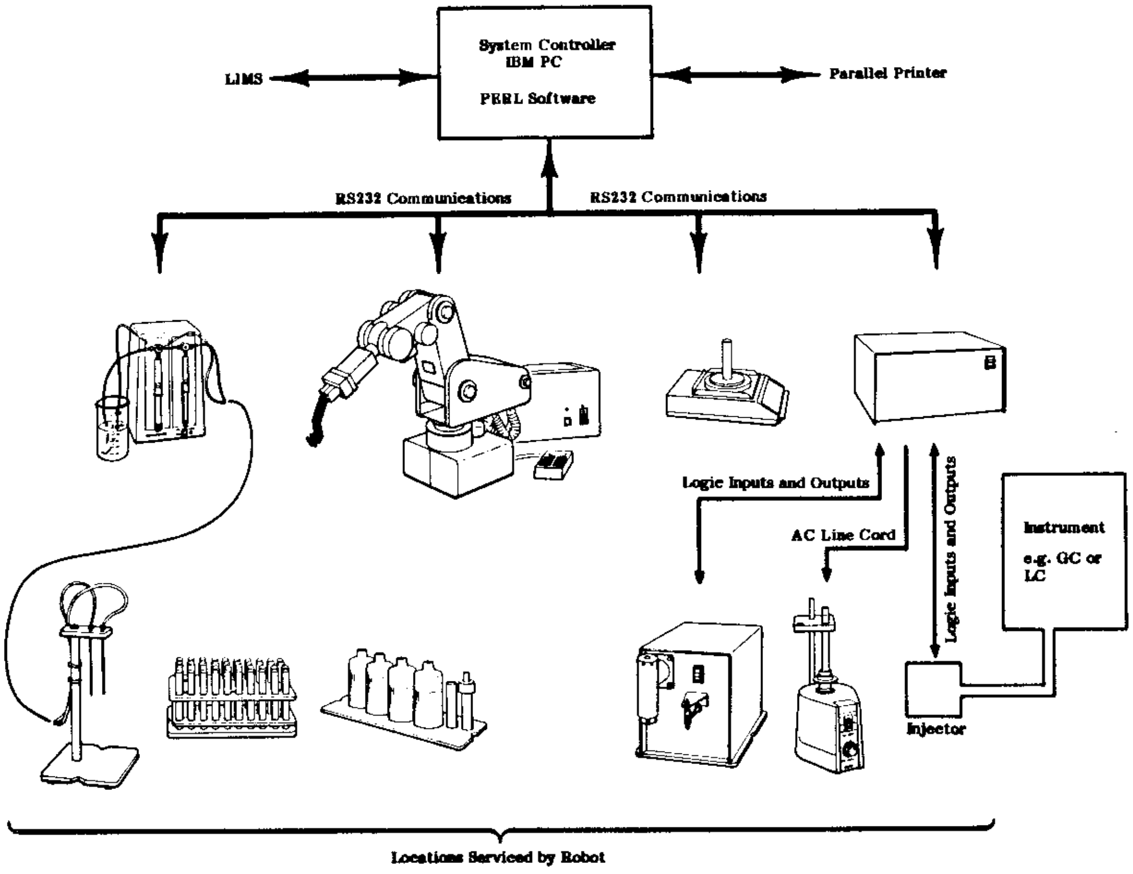


Figure 1-1 - Interaction Among Modules of The MasterLab Automated Sample Preparation System

BASIC statements, the architecture of the programs is also similar.

You will accumulate a certain amount of data (weights, volumes, numbers of samples) during sample preparations. PERL is capable of processing this data. In fact, PERL can be used for any usual programming needs; the difference from other languages lies mainly in its additional ability to communicate with robots and other devices easily. It was created especially for performing physical tasks using laboratory robotics.

The PERL Programming Environment

The PERL programming environment (see Figure 1-2) consists of a set of programming tools to help you develop PERL procedures. Some of these tools you will never use directly. Those with which you do interact are menu-driven for ease of use.

As can be seen in Figure 1-2, you will use the System Configuration Utility to store the configuration of the system in the system configuration file,

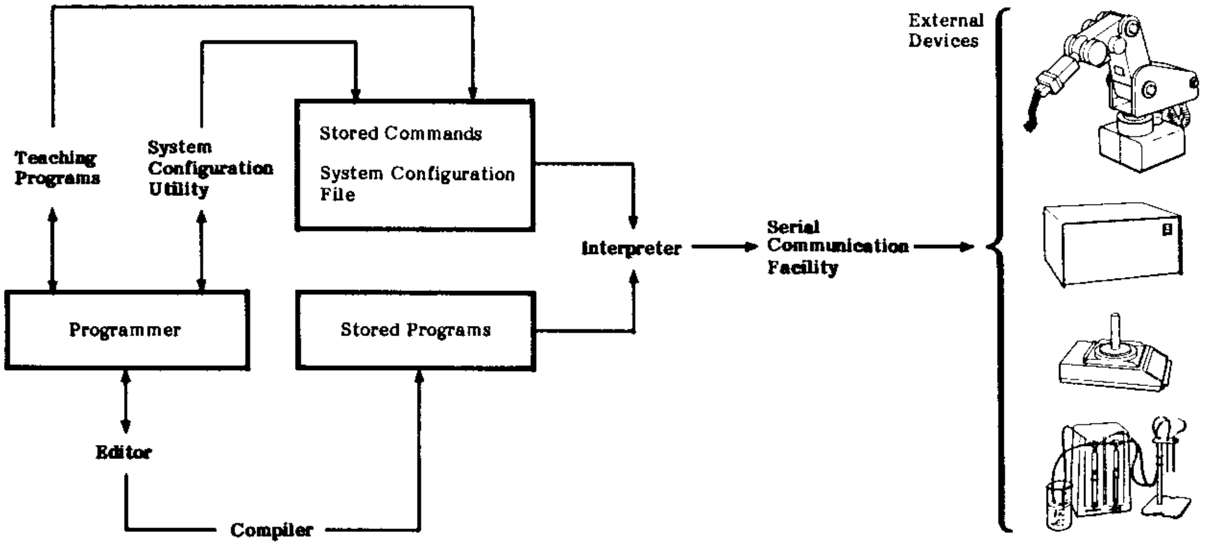


Figure 1-2 - The PERL Programming Environment

and the Teach Programs to create new stored commands for controlling the system modules. You will also use the Editor to create PERL procedures, which are compiled by the Compiler and stored.

When the stored procedure is run, the Interpreter uses both the stored commands and the system configuration file to interpret the procedure statements. Communication during procedure execution is handled through the Serial Communication Facility.

The following paragraphs describe more fully the individual components of the PERL environment.

Tools Used Directly by the Programmer

- The System Configuration Utility is an interactive program which allows you to describe the configuration of your system. It creates a system configuration file (PERL.SCF) which indicates what devices are at what ports, the initialization parameters for the ports and devices, communication protocols, and the spelling of system commands. The computer reads this file each time PERL is loaded into memory, and it sends the appropriate initialization commands.

  Once your system has been installed, you need not use the System Configuration Utility again unless you add, delete, or move a device. In that case, change the system configuration file. Your programs do not need to be changed, since all device communication is based on the system configuration file.

- Teach Programs are interactive programs that allow you to give names to various positions, actions, and quantities which will be needed in your laboratory. In teaching the robot, for example, you use the Teach Pendant, provided with the robot, to move it to the desired position. You then use the Teach Program to name the position. Both the name and the corresponding position are stored on disk, ready to be used as commands in PERL procedures. Note that you do not have to calculate the position; the position parameters are determined by the system. If you change a location later, you can use the Teach Program to change the stored parameters. You need make no changes in PERL procedures.

- The PERL Editor is used to create and edit PERL procedures. It is a full screen editor which takes advantage of the function keys and other special keyboard keys available on the System Controller (IBM PC). As a result, it is both powerful and easy to use. The completed procedure is automatically compiled when you save it.

Tools Invisible to the Programmer

- The Serial Communication Facility allows communication with external devices via RS232 ports. It is the basis of all communication between the MasterLab software and external devices.

- The Compiler is embedded in the Editor. After you create a procedure, it is automatically compiled into an intermediate form before it is stored on disk. This decreases run-time when the procedure is executed.

- The Interpreter interprets the statements of the compiled procedure in sequence when you run it. As each statement is interpreted, the appropriate action occurs.

The Structure of PERL Applications Programs

A PERL procedure, like any other, consists of a series of statements. As an example, consider the procedure below. (PERL does not use line numbers, and they are included here only for reference.)

Ref. No.	Program Lines	Comments
1	procedure demoprog	Procedure statement begins the procedure.
2	dim weight(40)	Dimensions array variable i.
3	for i = 1 to 25	Line 3 begins an action repeated 25 times. The end of this is at line 21.
4	test_tubes i	Removes a tube from location i in the rack named "test_tubes".
5	parallel	Lines 5-8 allow two devices to act simultaneously.
6	go_to_syringe	
7	fill_syr_5ml	
8	end parallel	
9	dispense_5ml	Lines 9-11 (also 4,6,7,14,15,17,20) are commands created with the Teach Programs and stored on disk.
10	go_to_mixer	
11	mixer_on	
12	set timer 1 for 15 seconds	Lines 12 and 13 suspend operation for 15 seconds.
13	wait for timer 1	
14	mixer_off	
15	weight(i) = weigh_sample	
16	if wt(i) <0.8 then	Lines 16-20 contain a conditional statement and an alternative action if the condition is true.
17	discd_tb	
18	else	
19	proc_smp	Calls a subroutine to be executed.
20	end if	
21	next i	End of repeated action begun at line 3.
22	end procedure	Last line of procedure.

As you can see from the example, a PERL procedure can look quite different from programs in other languages. This is because of the unique user-created commands, such as "weigh_sample". There are also some logical constructs (such as the "parallel", for example) which are provided to meet the unique needs of robotics. Much of the logic of the procedure, however, is similar to BASIC, so that the overall architecture will be familiar.

Note: Command names may contain up to 16 characters, the first of which must be alphabetic. Spaces are not permitted in the user-created commands. This is the reason for the presence of the underline character ("_") in most such commands. You must not use in command names either decimal points or symbols which the system could interpret as arithmetic operators (+, -, /, *).

PERL applications procedures are usually constructed from shorter procedures. Both the procedures and the subprocedures begin with "procedure" statements. Section 6 of this manual will take you through the steps in developing an applications procedure from subroutines.

Creating a PERL Procedure

When the MasterLab System is installed, or whenever you add or change a module, you must use the System Configuration Utility, "config". See "Creating the System Configuration File" in Section 2. Assuming that this has been done and the system configuration file is complete, you can create PERL procedures as follows:

1) Use the Teach Programs to create the unique statements which will be needed in the procedure. Examples might be "fill_syr_2ml" or "move_to_crimper". Use of the Teach Programs is described in Section 3.

2) Access the PERL Editor as described in Section 5.

3) Using the BASIC-like PERL statements in Section 4 to provide the overall procedure structure, and your created statements from step 1 to give commands for specific actions, enter your procedure and edit it.

4) Save the procedure.

Note: If your software contains the optional Debug program, press the F5 (Debug) function key to use it.

The following section explains how you can use the various sections of this manual as aids to PERL programming.

USING THIS MANUAL

Organization

The sections of this manual are arranged in the order in which you would logically need them in order to write a PERL procedure:

Section 2, System Commands, describes how to load PERL, how to use the function keys available from the main PERL screen, and what commands can be issued from the keyboard.

Section 3, Teach Programs, contains instructions for using the Teach Pendant and the interactive Teach Programs to create the commands needed for your new applications.

Section 4, The PERL Language, contains descriptions of the statements which are permanent features of PERL.

Section 5, The Editor, explains how to access and use the Editor for writing and editing procedures and files. It includes a tutorial section.

Section 6, Creating and Running an Application Program, takes you through the steps involved in planning and writing an applications program. It includes sample subroutines for common operations.

Conventions

PERL statements or portions of statements (such as "if a = b then" and "x = abs(y)" which are referred to within the text of the manual are in lower case, enclosed in quotation marks. This is not meant to imply that quotes are needed around statements when programming. No quotes are used in the case of procedure segments which are indented to set them off from the surrounding text:

```
if a = b then
    get_tube
else
    process_sample
end if
```

Responses which you must type in after a prompt are printed in bold type:

name: **newrobot**

SEND US YOUR IDEAS

We at Perkin-Elmer are continuing to work to improve the features of PERL, and to ensure that the software performs as well as possible. We welcome your suggestions for enhancements to PERL, as well as your reports of problems with it.

Please use the Software Change Request form, found at the beginning of this manual, to communicate with us.

SOFTWARE LICENSING AGREEMENT

When you unpack your system, you will see that the PERL disks are sealed. The seal constitutes a self-executing license for the software contained on the disk. Breaking the seal indicates acceptance of the terms of the Software License Agreement located in the front of this manual. A notice on the outside of the disks reads as follows:

NOTICE

Program License Agreement

The Programs Herein Are Licensed To You For Use On A Single Computer Under The Terms And Conditions Stated In The License Found In The Front Of Your Operator's Manual. Opening This Package Indicates Your Acceptance Of Those Terms And Conditions. If You Do Not Agree To Those Terms And Conditions, You May Return This Package <u>Unopened</u> For A Full Refund Of The License Fee.

2 SYSTEM COMMANDS

UPDATING YOUR PERL DIRECTORY

TO VERSION 1.60

If you have a PERL Directory (PERL.DIR file) containing commands created with a version of PERL earlier than 1.60, you will need to convert it before using it with PERL 1.60. To do this, run the program XLATE, which is included with your new PERL disks.

> **Important:** If you try to use your existing directory with the new software before translating it, your directory of commands may be corrupted and lost.

Use the following procedure:

1) In drive B of the System Controller, place your work disk containing the files PERL.DIR and ROBOT.999. (These are the files to be translated from an earlier version to PERL 1.60.)

2) In drive A place the disk containing the XLATE program.

3) To make drive B the default drive, type

   b:

   and press Enter. The prompt will become B>.

4) To run the XLATE program, type

   a:xlate

   and press Enter.

   The screen will prompt you to enter the name of your robot.

5) Type the name your robot was given in your system configuration file, and press Enter.

6) If you have not inserted the disk with the ROBOT.999 file, the screen will prompt you to do so. When the disk is in place, press Enter.

   The screen will ask you to identify the version of PERL you used to make your directory.

7) Highlight either "PERL Directory 1.50" or "PERL Directory 1.58". Press Enter.

   The screen will ask:

   > Do you have syringe commands (y/n)?

8) Type **y** if you have syringe commands in your directory; type **n** if you do not.

   If you answered yes, the screen will prompt you to enter the name of the syringe.

9) Type the name given to the syringe in your system configuration file. Press Enter.

   > **Important:** If you have more than one syringe, you will lose some commands when the directory is translated. To minimize the number lost, give the name of the syringe for which you have the most commands.

   The screen will list the model numbers of the available syringes.

10) Highlight the model number of the syringe you named in step 9. Press Enter.

11) If you have not inserted the disk with the PERL.DIR file, the screen will prompt you to do so. When the disk is in place, press Enter.

    The system will now execute the conversion to PERL version 1.60 as follows:

    * It creates a new PERL.DIR file which contains your commands in a form compatible with PERL version 1.60.

    * It places your old directory (version 1.50 or 1.58) in a back-up file called PERL15X.DIR.

    * It converts the ROBOT.999 file to a file called ROBOT.HOM.

> **Important:** The structure of the system configuration file is different in PERL 1.60. Before using your new software, reconfigure your system by running the CONFIG program that came with PERL version 1.60. See "Creating the System Configuration File" in Section 2 of your PERL Instructions.

2 SYSTEM COMMANDS

SETTING UP YOUR PERL DISKS

Backing up DOS and PERL Disks (One Time Only)

You should make a back-up copies of your PERL disks for everyday use, so that your master copy can be stored. You will find it a convenience to have a copy of the most frequently used DOS commands on the same diskette that contains PERL. The following procedure copies both DOS and PERL onto a single disk. After you have made this disk, use it with the procedure under "Loading the PERL Work Disk" whenever you wish to load PERL.

1) With power to the System Controller (IBM PC) turned off, insert the DOS diskette into drive A. Turn the Controller on. (If your System Controller is already on, insert the DOS diskette and then reset the Controller by pressing CTRL, ALT, and DEL simultaneously.)

2) When the date is displayed, press Enter. When the time is displayed, press Enter again. (You should set the date and time, but you will do so when you boot your new DOS/PERL diskette in the next section.)

3) When the DOS prompt (A>) appears, put a new, unformatted diskette in drive B.

4) Type the following command:

   format b:/s

   Press Enter, and wait for the DOS prompt to reappear.

5) When the system asks you "format another", insert a second blank disk in drive B, and type **y**.

   This will give you two formatted disks, on which you can copy your PERL program disk and your System Configuration Utility (CONFIG) disk.

6) Remove the DOS diskette from drive A, and insert your PERL program disk. Type

   copy a: perl *.exe b:

   and press Enter. The entire PERL diskette will be copied onto the diskette in drive B.

6) Remove the PERL diskette from drive A. Remove the diskette from drive B, and label it "DOS/PERL work disk".

7) Insert your CONFIG disk in drive A and your other formatted blank disk in drive B. Type

```
copy a: *.* b:
```

   and press Enter. The CONFIG disk will be copied onto the disk in drive B.

8) Remove the disk from drive B and label it "CONFIG work disk."

9) Remove the CONFIG disk from drive A. Store the original PERL and CONFIG disks where they will be safe from heat, magnetic fields, dirt, or mechanical injury.

Loading the PERL/DOS Work Disk

For this procedure, use the DOS/PERL work disk which you created in the preceding section.

1) First load DOS. To do this, with power to the System Controller turned off, insert your DOS/PERL diskette into drive A. Turn on all external devices connected to your System Controller, then turn the Controller on. (If your System Controller is already on, insert the diskette and then reset the Controller by pressing CTRL, ALT, and DEL simultaneously.)

2) After a pause, the date will be displayed. If it is not correct, enter the correct date in either of these formats:

```
8-1-85
8/1/85
```

   Press Enter.

3) The present time will then be displayed (hours, minutes, seconds, hundredths of a second). If it is not correct, enter the correct hours and minutes in the format

```
9:05
```

Press Enter. The DOS prompt (A>) will appear.

Creating a Data Disk

In general, you will load the PERL/DOS work disk into drive A, and use a data disk in drive B for saving your created procedures, directory of commands, and data files.

To create your first data disk, use the following procedure to place the necessary files on a disk in drive B:

a) With the PERL/DOS work disk in drive A, place a formatted disk in drive B.

b) Type

copy *.HOM b:

and press Enter. The system will copy the initialization procedure(s) for your robot(s) onto the disk in drive B.

c) To make drive B the default drive, type **b:**, and press Enter. The prompt will become B>.

d) Place your CONFIG work disk in drive A. Create your system configuration file (PERL.SCF) as follows. Type

a:config

and press Enter. Follow the procedure given in "Creating the System Configuration File", below. The system will place the file PERL.SCF on the disk in drive B.

e) Load PERL as described in "Loading PERL", below. Commands you create with PERL will be filed on disk B in PERL.DIR.

To create additional data disks (after you have a System Configuration File and a Directory of commands), proceed as follows:

a) With the DOS prompt (A>) displayed, place a previous data disk in drive A and a new, formatted disk in drive B.

b) Copy the files you will need onto the new data disk by typing the following series of commands. Press Enter after each command.

```
copy *.HOM b:
copy PERL.SCF b:
copy PERL.DIR b:
```

c) Load PERL as described in "Loading PERL", below.

CREATING THE SYSTEM CONFIGURATION FILE

Introduction

Before you can operate the MasterLab System, you must configure the software for the particular modules you have in your system. "CONFIG" is an interactive program which allows you to create a file (PERL.SCF) that identifies the communication requirements of the modules included in your system. You can identify the following:

* Accessory modules in the system
* Port location of each module
* Initialization parameters for each module
* Communication protocol for each module

The system reads the system configuration file each time you load PERL into memory. This results in the initialization of the communication ports on the IBM PC and of the modules on the system.

Once the MasterLab System has been installed, you need not use CONFIG again unless you add, delete, or change a module. In that case, you would change the system configuration file to reflect the changes to the system. For example, if you added a bar code station to the system, you would run CONFIG in order to make the necessary addition to the system configuration file.

Loading Config

1) Load DOS as described in "Loading the PERL/DOS Work Disk", above.

2) Place the CONFIG disk in drive A and a data disk in drive B. With the DOS prompt (A>) displayed, type **b:** and press return. This will make the data disk in B the destination on which the system will save your configuration file (PERL.SCF).

3) To load the Config program from disk A into memory, type

a:config

and press Enter.

The display shows the main menu of Config. Four options are available:

- Add a Module
- Change a Module
- Delete a Module
- Display Module Information

The bottom of the display identifies the tasks assigned to each of the five function keys which are active in Config. The key assignments are:

Key	Label	Function
F6	Help	Displays information on the operation of various parts of the program.

Note: The HELP function has not yet been implemented.

Key	Label	Function
F7	Quit	Stops the program and returns to the DOS environment.
F8	Bkup	Returns to the menu displayed prior to the selection of the current menu.
F9	Up	Highlights the menu item on the line above the item currently highlighted.
F10	Dn	Highlights the menu item on the line below the item currently highlighted.

Adding a Module

Every module that is installed with your MasterLab System, or is added to the system later, must be identified in the system configuration file. This enables the System Controller to communicate with the module. Using Config to add a module (for example, a printer or a syringe) to the MasterLab System sets the communications parameters associated with the new module. Either the system sets the parameters by default, or you select them.

The following is a general procedure for adding a module to the system. The procedure for adding a robot to your MasterLab System is described in Section 3 of the MasterLab System Instructions. You will find the procedure for adding each accessory module in the manual provided with the module.

To add a module to the system, proceed as follows:

1) Use the Up (F9) and Dn (F10) function keys (or the up and down arrow keys) to highlight the option labeled:

    Add a Module

   and press the Enter key.

   The display now lists categories of modules which can be added to the system. The choices are:

    Robot
    Printer
    Syringe
    Balance
    Bar Code Reader
    Device Interface
    Capping Station
    All Other Modules

2) Highlight the type of module you wish to add, and press the Enter key.

   Note: If the module is a syringe, the system will list the available models: 9030, 9040, 9050, 9060. Highlight the model number of the syringe you are adding, and press the Enter key.

   If the module is a balance, the system will ask you to indicate whether it is a Mettler or a Sartorius.

   The system now asks you to enter the communications requirements for the new module.

3) If you selected Printer as the module to be added, the system prompts:

    Port Type

   and two options, RS-232 and Centronics, are available.

   Highlight the option which corresponds to the interface on the printer you are adding, and press Enter.

   Note: For devices other than printers, the system assumes an RS-232 serial interface.

4) a) For RS-232 devices:

When the display prompts:

Port Mnemonic

highlight the name of the communications port on the System Controller to which you connected the module, and press the Enter key. For example, highlighting

COM3:

and pressing Enter identifies communications port 3 on the System Controller as the port to which you have connected the module.

Note: COM3: through COM6: are available on the standard System Controller. COM1:, COM2: and COM7: through COM10: are optional ports.

b) For Centronics interface printers only:

When the display prompts:

Port Mnemonic

highlight the name of the Centronics printer port to which the printer is connected. For example, if the printer is connected to the primary parallel interface port, highlight:

LPT1:

and press the Enter key.

Note: LPT1: is usually used when the System Controller contains a Hercules graphics card.

Important: Steps 5 through 10 establish parameters which are defaulted for standard MasterLab System modules. If the type of module you are adding was listed on the Config "Add a Module" menu, proceed to step 11. If you entered "All Other Modules" at step 1, refer to the manual provided with the module to determine its communications requirements.

5) The next prompt displayed is:

Handshaking

and two options, Yes and No, are available.

Highlight YES if the module supports hardware handshaking through the Clear to Send (CTS), Data Terminal Ready (DTR), Data Set Ready (DSR), and Request to Send (RTS) lines.

Highlight NO if the handshake is not supported.

Press Enter to complete this selection.

6) The system next requests you to select the baud rate at which the module receives and transmits data. The choices range from 50 to 9600 baud. Highlight the value at which the module receives and transmits data, and press Enter.

7) The system prompts for a parity selection. The three options are:

   No Parity
   Odd Parity
   Even Parity

   Highlight the option used by the module, and press the Enter key.

8) The system prompts for the number of stop bits; that is, the number of bits terminating each character sent. The options are:

   One Stop Bit
   Two Stop Bits

   Highlight the option required by the module, and press the Enter key.

9) The system prompts for the number of data bits transmitted by the module. The options available are:

   Five Data Bits
   Six Data Bits
   Seven Data Bits
   Eight Data Bits

   Highlight the option required by the module, and press the Enter key.

10) The system next prompts for the timeout value; that is, the time (in seconds) that the computer waits for the module to respond to a command. If no response has been received when the time has elapsed, the computer displays an appropriate error message.

    When the system prompts:

    Timeout

    enter a value up to 99 seconds, and press Enter.

    Note: The timeout function has not yet been implemented, so the system will make no use of your response to this prompt.

11) The system then prompts:

    Module Name

Enter a name, and press the Enter key. Names must begin with an alphabetic character, may include up to sixteen characters, and may not include spaces. Spaces can be indicated using the underscore character. The system will use the name to identify the module in the PERL environment.

Notes:
1. The following characters should not be used as names in SCU or PERL.

*	+	%
/	-	!
@	$	

2. If you are entering the name of a syringe, the system will continue to prompt you for more names. If you have syringes "daisy chained", with more than one connected to one port, enter the name of each syringe at the port, pressing Enter after each one. When all are entered, press Enter again without typing a name. Proceed to step 12.

Important: Steps 12 through 14 establish parameters which are defaulted for standard MasterLab System modules. If the type of module you are adding was listed on the Config "Add a Module" menu, proceed to step 15. If you entered "All Other Modules" at step 1, refer to the manual provided with the module to determine its communications requirements.

12) The system prompts for the Initialization Sequence. The initialization sequence is the action or series of actions that sets the module to its starting position. It can also request a response from the module before initialization proceeds. If a response is required from the module, the following command is included:

,?xxx, (where xxx is the required response)

You may enter any ASCII character as part of an initialization sequence. For example, if the initialization were for the System Controller to query "Hello, how are you?", and the module to return "I am fine", the initialization sequence would be:

```
Hello<,> how are you<?>,?I am fine
```

The angle brackets around the first comma and question mark indicate that they are included in the query, not part of a command. It would be equally correct to use

```
<Hello, how are you?>,?I am fine
```

Angle brackets are also used with an ASCII command such as <CR> (carriage return). See the table of ASCII and PERL abbreviations for control characters, found in Appendix 3 of this manual.

Typical Initialization Sequences for two modules are shown below. (Since these are standard modules, their initialization sequence would be defaulted for you.)

Module	Initialization Sequence	Explanation
Printer	<FF> <CR>	Form feed followed by a carriage return.
Device Interface	C,?*	Clear module and wait for asterisk to be returned.

Enter the combination of ASCII and module commands that defines the sequence for your module, and press Enter.

13) The system prompts for the Input Terminator, the ASCII character or sequence of characters used to terminate the response coming from the module to the System Controller (IBM PC). For example, a typical input terminator is a carriage return, <CR>, or a carriage return, line feed, <CR> <LF>.

    Enter the ASCII character or sequence of characters for the desired input terminator in angle brackets < >, and press the Enter key.

14) The system prompts for the Output Terminator, the ASCII character, or the sequence of characters used to terminate a command sent to the module from the System Controller (IBM PC). For example, a typical output terminator for a printer is a carriage return followed by a linefeed; that is, <CR> <LF>.

    Enter the ASCII character for the desired terminator in angle brackets < >, and press the Enter key.

15) To store the information just entered for the module on the PERL diskette and return to the DOS environment, press the Quit function key (F7).

    The system queries:

        Save the current modifications? (y/n)

    Press:

        y - to save the communications options for the module

        n - to abort the storage operation

    If you type **y**, the communications parameters selected for the module will be stored on your data disk in the system configuration file named PERL.SCF.

Changing a Module

With Config you can change the communications parameters assigned to any module on the system.

To change a module, proceed as follows:

1) Use the Up and Dn keys (F9 and F10) to highlight the option on the main menu of Config labeled:

   Change a Module

   and press the Enter key.

   A list of all the modules presently in the system configuration file is displayed on the screen.

2) Highlight the name of the module whose communications parameters you wish to change, and press the Enter key.

   The present parameter settings for the device are displayed on the screen (see example below).

```
Port Type          RS-232
Port Mnemonic      COM7:
Handshaking        Yes
Baud Rate          9600
Parity             Even
Stop Bits           1
Data Bits           7
Port Timeout       10 seconds


Module Name        Instrument
Module Type        Other Device
Initialized        On Start-up
Init. Sequence
Input Terminator   <CR> <LF>
Output Terminator  <CR>
```

3) Use the Dn function key (F10) to move the cursor to the first parameter that you wish to change.

4) Press the Enter key to delete the current entry for the parameter.

5) a) If the parameter requires a typed entry, type the desired entry and press the Enter key.

   b) If the parameter requires a selection from items in a menu, use the Up and Dn keys (F9 and F10) to highlight the desired item, and press the Enter key to complete the selection.

6) When the parameters for the module are modified as you wish, press the Dn function key (F10) until you reach the bottom of the parameter list. You will see the following message:

   Please Press the ENTER Key to Continue

7) To continue operation in Config, press the Enter key.

8) To store the information just entered in the system configuration file and return to the DOS environment, press the Quit function key (F7). The system queries:

   Save the current modifications? (y/n):

   Press:

   y - to save the modified communications parameters for the module(s)

   n - to retain the previous set of communications parameters for the module(s)

Deleting a Module

Config enables you to delete any module from the MasterLab System. Proceed as follows:

1) Use the Up and Dn keys (F9 and F10) to highlight the option on the main menu of Config labeled:

   Delete a Module

   and press the Enter key.

   A list of the modules presently in the system configuration file is displayed on the screen.

2) Highlight the module that you wish to delete, and press the Enter key. The module is deleted from the system configuration file, PERL.SCF.

Important: Deletion is instantaneous. Be certain that you wish to delete the module from the file before you press the Enter key. If you make a mistake, use the Add a Module option to return the module to the file.

3) To store the modified system configuration file on your data disk and return to the DOS environment, press the Quit function key (F7).

   The system queries:

   ```
   Save the current modifications? (y/n):
   ```

   Press:

   y - to save the modified file

   n - to retain the previous set of modules

Displaying Module Information

To display the communications parameters for any module that is already present on the system, proceed as follows:

1) Highlight the option labeled:

   Display Module Information

   and press the Enter key.

   A list of the modules which are configured for use with the MasterLab System appears on the screen.

2) To display information for any of the modules listed, highlight the module name and press the Enter key. A display similar to that shown below appears on the screen.

```
Module Information          PERL Configuration Utility V1.0

Port Type                   RS-232
Port Mnemonic               COM1:
Handshaking                 Yes
Baud Rate                   9600
Parity                      Even
Stop Bits                   1
Data Bits                   7
Port Timeout                10 seconds

Module Name                 ROBOT1
Module Type                 Robot
Initialized                 On Start-up
Init. Sequence
Input Terminator            <CR>
Output Teminator            <CR> <LF>
```

Refer to "Adding a Module", above, for information about the individual items in this list.

3) To display information for the other modules on the system:

   a) Press the Enter key to redisplay the list of modules presently in the system.

   b) Highlight the name of the desired module, and press the Enter key.

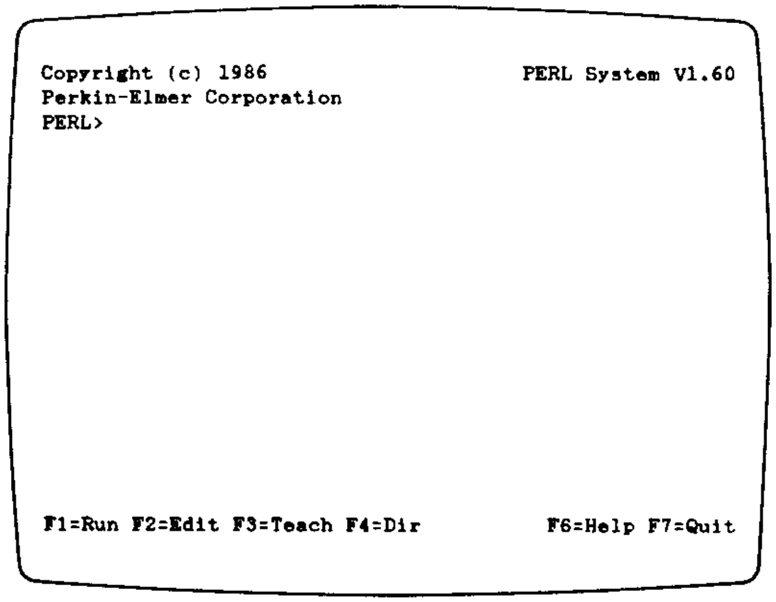
4) If you need to make changes in any parameters, use the "Change a Module" option.

LOADING PERL

1) Load the PERL/DOS work disk as described earlier.

2) With the DOS prompt displayed, place your data disk in drive B. Type **b:** and press Enter to make drive B the default destination for the files you create with PERL.

3) Type **perl** (or the name of the batch file you have created; see next page), and press Enter. The MasterLab devices which initialize at Start-up will initialize, as requested by the system configuration file. The PERL main screen (see Figure 2-1) will appear. When this screen is displayed, the PERL direct command processor is in control; that is, any PERL command typed in will be executed. In addition, the 10 function keys described under "Function Keys Available on the PERL Main Screen" are active.



```
Copyright (c) 1986                          PERL System V1.60
Perkin-Elmer Corporation
PERL>













F1=Run F2=Edit F3=Teach F4=Dir                F6=Help F7=Quit
```

Figure 2-1 - PERL Main Screen

Default Conditions at Start-up

The PERL default conditions are set in the file PERL.BAT, which runs automatically when you load PERL. A listing of this file appears similar to the following:

```
echo off
verify on
perl =10000 -d3000 -p6000 -rtest
```

[illegible] spaces in line 3. They are critical!!

The third line of the file allocates the computer memory to be used for stack space, directory, and program. In the example above, the stack space is 10,000 bytes, the directory size 3000, and the program size 6000. You can allocate all the available memory in your computer, if you want, except for approximately 310K for PERL itself (version 1.60).

In addition to the -d and -p flags for setting directory and program space, you can set several other flags in PERL.BAT to modify start-up conditions. In the example above, the -r flag indicates that a procedure called "test" is to run immediately after start-up. Table 2-1 lists the available flags.

If you want to change the default conditions, load PERL.BAT into the PERL Editor (see Section 5). Use the Editor to change the flags in the third line of the file.

TABLE 2-1
FLAGS WHICH MODIFY PERL START-UP CONDITIONS

Flag	Explanation	Example
c	Execute a command immediately.	-cdisplay devices
d	Set directory size.	-d16
i	Set standard input device. Defaults to console (con:).	-ioldfile
l	Set device in which log is kept. Defaults to console (con:).	-lthursfile.log
o	Set standard output device. Defaults to console (con:).	-oprinter
p	Set program size.	-p6000
r	Run a procedure immediately.	-rrobt_tst
None	Set stack size.	perl =1000

THE DIRECT COMMAND PROCESSOR

The direct command processor (DCP) is in control whenever the PERL> prompt is displayed. It allows you to execute any commands that you have defined or that are a part of PERL. Simply type in a command or a procedure name and press Enter.

The following features are available to you from the DCP:

Feature	Section in which Described
pre-defined commands	2
screen editing commands (not part of the PERL Editor)	2
commands controlled by function keys	2
PERL Directory	2
Teach Programs	3
PERL Editor	5
execution of user-defined commands	2

PRE-DEFINED COMMANDS

The commands described in this section may be issued from the keyboard as described above, or they may be used in PERL applications programs where appropriate.

Command	Function	Example
DEVICE COMMANDS:		
init	Initializes specific devices by the name entered in "CONFIG".	init robot_1
use	Specifies by name the device to be used when there are multiple devices of the same type in the system.	use robot_2
display devices	Displays the configurations for all devices in the System Configuration File. From this display, press F8(Backup) to return to the main PERL screen.	display devices

DIRECTORY COMMANDS:

load	Read Directory from disk. This causes any new Directory entries in memory to be lost. The Directory <u>which is loaded</u> is automatically backed up in the file PERLDIR.BAK.	load
save	Write Directory to disk. This saves any additions to the Directory which are in memory, placing them in the file PERL.DIR.	save

I/O COMMANDS:

redirect	Changes the standard input or output device. (Default input device is the keyboard. Default output device is the monitor screen.) Available from DCP only.	redirect output to com1: redirect input from newfile
log	Creates a file on disk which will contain everything sent to the screen.	log monday.log
clear	Clears the PERL screen.	clear

ROBOT COMMANDS:

open	Open grippers.	open
close	Close grippers.	close
speed	Set speed of robot. (0 - 9 possible; 0 slowest, 9 fastest)	speed 7
up	Moves the robot up a specified distance (in mm) from its present position.	up 5
down	Moves the robot down a specified distance (in mm) from its present position.	down 4

relative	Moves the robot relative to its present position. The arguments specify mm in the x, y, and z directions and degrees of wrist pitch and roll, respectively. (See "Robot Commands" in Section 4 for details.)	relative 2,4,6,5,5 relative 2,2,4 relative 0,0,0,20

TIMER COMMANDS:

set timer	Sets one of ten software timers for specified number of minutes or seconds.	set timer 1 for 10 seconds
wait for timer	Tells system to do nothing until previously-set timer has elapsed.	wait for timer 1

SYSTEM CONTROL COMMANDS:

dos	Returns to operating system. PERL remains in memory. (See "System Control Commands" in Section 4 for other uses of "dos".)	dos
system	Returns to operating system. PERL is aborted.	system
link	Links all the subprocedures used in a main procedure. Generates an executable disk file with the extension ".LNK" appended to the filename.	link proc_mix

DATE AND TIME COMMANDS:

date	Displays date.	date
day	Displays the day of the week.	day
month	Displays the month.	month
time	Displays time.	time

THE DCP SCREEN EDITOR

The screen editor embedded in the DCP makes use of the editing keys of the System Controller (IBM PC). It allows you to edit command lines and to move the cursor anywhere on the screen. You may also return to a previously-issued command and execute it again; simply move the cursor to the end of the command line and press Enter. The functions of the editing keys are as follows:

Key	Function
Up arrow	Moves the cursor up one line.
Down arrow	Moves the cursor down one line.
Right arrow	Moves the cursor one character to the right.
Left arrow	Moves the cursor one character to the left.
Home	Moves the cursor to the beginning of its present line.
End	Moves the cursor to the end of its present line.
PgUp	Moves the cursor to the top of the screen.
PgDn	Moves the cursor to the bottom of the screen.
Ins	Inserts all characters typed subsequently into the position to the left of the cursor. Press Ins again to return to normal operation.
Del	Deletes the character which is at the present cursor position.
any character key	Overwrites the character which is at the current cursor position.
Tab	Moves the cursor to the next tab position.

FUNCTION KEYS AVAILABLE ON THE PERL MAIN SCREEN

The PERL main screen assigns functions to several of the function keys on the left side of the keyboard. This section will explain the use of each one.

Key	Function of Command
F1=Run	Runs a PERL applications procedure. When you press F1, you will receive the message "Please enter procedure name". Type in the name and press Enter.
F2=Edit	Accesses the PERL Screen Editor, which is used to write and edit procedures and information files. The use of the Screen Editor is described in Section 5 of this manual.

F3=Teach	Accesses the Teach programs for the robot and other modules of the MasterLab System which are capable of RS232 communication. These programs are used to create the commands necessary to the operation of these modules. The use of the Teach programs is explained in Section 3 of this manual.
F4=Dir	Accesses the PERL Directory, which contains a list of all PERL files and procedures, as well as all commands which you have taught to the system by using the Teach programs. Use of the Directory is explained in Section the preceding section.
F6=Help	Displays appropriate available help screens.

Note: The HELP function has not yet been implemented.

F7=Quit	Aborts PERL and returns you to DOS. This is the same as the "system" command described under "Leaving PERL".

Note: During execution of a PERL procedure, F7 changes and F8 becomes available, as follows:

F7=Halt	Stops procedure execution at the end of the present line.
F8=Continue	Resumes previously halted procedure, beginning at the next line.

THE PERL DIRECTORY

PERL will store in the PERL Directory (the file PERL.DIR) all the commands that you create with the Teach Programs. When you load PERL, the System Controller reads this file into memory, so that all the commands you have created are available.

The computer memory space allocated to the Directory is set in the batch file PERL.BAT. If your Directory becomes larger than the allocated memory, you will receive a screen message. You can then increase the Directory space, as described in "Default Conditions at Start-up", earlier in this section.

Each time you attempt to save your Directory on disk, PERL will verify that the disk contains enough free space. If you receive a message that there is insufficient space, insert another disk into the drive, and save the Directory again.

The Directory functions not only as the repository for the commands you create. You can also make use of it directly, in two ways:

* as a reference tool, in which you can check on the names you have given to commands;
* as a means by which you can execute commands without having to type them.

1) To see the PERL Directory, press F4 (Dir) from the PERL main screen (DCP). The Module Command Directory main screen will appear. It lists the types of modules you have connected to your system.

2) Use the arrow keys or F9 and F10 to highlight the module whose commands you wish to see. Press Enter.

   If the module you selected has several types of possible commands, then a menu of command types will appear.

3) If necessary, highlight the desired type of command, and press ENTER.

   The options menu for the module you selected will appear. In all cases, it lists the following options:

   Display/Execute Module Command
   Change Module Command
   Delete Module Command

4) The Display/Execute option is highlighted. Press Enter to select it. A display of commands you have created for the chosen module will appear (e.g. robot commands or syringe commands, depending on what module and command type you chose in steps 2 and 3).

5) Use the arrow keys to move the cursor to the command you wish to execute. If you have more than one screen of commands, use the PgUp and PgDn keys to move from screen to screen. Press Enter. (If you do not wish to execute a command, proceed to step 7.)

6) The system will execute the command and return you to the options menu for the module you selected.

7) Press F8 (Bkup) repeatedly to return to earlier screens, or press F7 (Quit) to return directly to PERL.

LEAVING PERL

When you wish to use DOS commands or load other software, you must leave the PERL environment. There are two ways to accomplish this:

- To return to the DOS Operating System while keeping PERL in memory, type **dos** and press Enter. You can then issue DOS commands. (For some commands, you may need to insert the DOS disk in drive A.) When you are ready to return to PERL, type **perl** and press Enter. The PERL screen reappears without the modules having to be re-initialized.

  Note: Your system must have sufficient memory in order to be able to use the DOS mode within PERL, and you must have copied the DOS file COMMAND.COM onto your PERL/DOS work disk.

- To abort PERL and return to DOS, type **system** and press Enter, or use F7 (Quit). You will then be able to issue DOS commands or to load other software.

3 THE TEACH PROGRAMS

3 THE TEACH PROGRAMS

INTRODUCTION

The feature which distinguishes PERL from other software languages is its ability to acquire new commands. Using the PERL Teach Programs, you can create commands for later use in applications programs. You can tailor these commands to the particular sample preparation tasks being carried out in your laboratory. PERL's ability to direct external devices to perform a variety of tasks is one reason why it is an advanced language.

Teach Programs exist for each of the programmable modules of the MasterLab System: the robot, the device interface, the syringe, the bar code reader, and the balance. The remainder of this introductory section explains what must be done before you can use the Teach Programs, and describes in general the interactive teaching process they employ. The major portion of Section 3 contains specific instructions for using four of the Teach Programs presently available. You will also find in the manual for each accessory module the Teach Program information relevant to that module.

Before the Teach Programs: The System Configuration File

Before the MasterLab System can be used, the System Configuration Utility must be used to create a system configuration file. (See Section 2.) The system configuration file contains the basic information the system must have in order for there to be communication among the various modules.

In the system configuration file, each module has been given a name by which it can be called in commands. If there is more than one module of a given kind (two syringes, for example), each one is assigned a separate name. The system configuration file contains information on how to initialize each module, its port, and relevant communications parameters.

If you have added a module to or deleted one from the system, be sure that you have changed the system configuration file accordingly before you attempt to use the Teach Programs. For example, if you add a balance to your system, the balance Teach Program will not be available to you until you have used the "Add a Module" program of the System Configuration Utility to inform the software about the new module.

The Interactive Teaching Process

A PERL applications program is, in large part, a series of communications to the various modules, telling them to change a present condition to a new condition. Table 3-1 gives examples of changes which some modules might undergo, and command names which you could use to initiate the changes.

TABLE 3-1
TYPICAL USER-CREATED COMMANDS

Module	Initial Condition	Final Condition	Created Command
robot	grippers over test tube rack	grippers under dispensing stand	go_to_syringe
device interface	AC outlet 1 OFF	AC outlet 1 ON	turn_on_mixer
balance	weight not being communicated	weight being communicated	weigh_sample
syringe	plunger at 0.0 ml	plunger at 5.0 ml	fill_syr_5ml

The Teach Programs are interactive. They allow you to put the module in each successive condition needed for your application, and then to assign a command name to that condition. The commands you create are stored in the PERL directory. From then on, when the go_to_syringe command from Table 3-1, for example, is entered from the keyboard or encountered in a running program, the robot will place its hand under the dispensing stand. (The underline character is present in many command names because no spaces are permitted.)

USING THE TEACH PROGRAMS

The Teach Programs are accessed from the PERL main menu screen by means of function key F3 (Teach).

Each Teach Program consists of a series of screens which present you with menus of available options or ask you to input information from the keyboard. To make choices on the menus, highlight the desired option, using the up and down arrow keys or the F9 (Up) and F10 (Dn) keys. Then press Enter.

The F8 (Bkup) function key, which is always available, allows you to return to previous screens.

When you press F3 to access the Teach Programs, a Teach Program selection menu similar to Figure 3-1 appears. It lists the kinds of modules for which your system is configured, for which Teach Programs are available to you. Highlight the kind of module you want to teach, and press Enter.

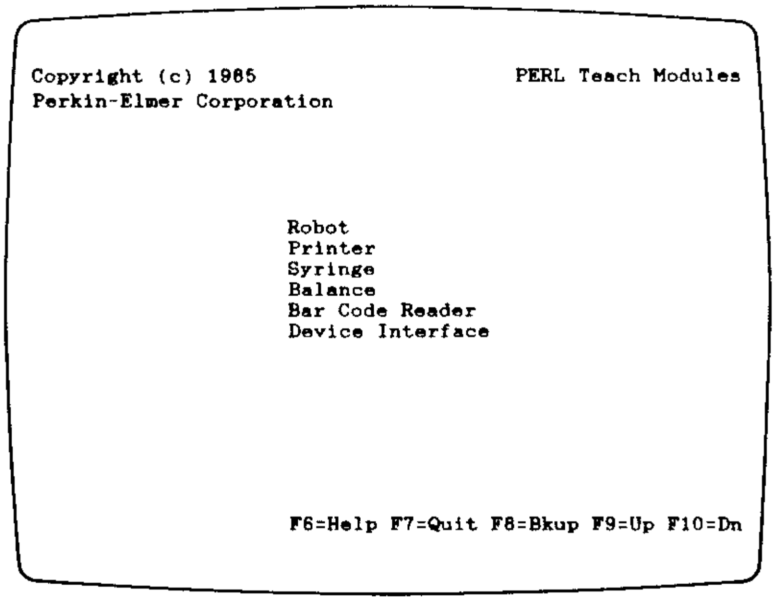

```
Copyright (c) 1985                              PERL Teach Modules
Perkin-Elmer Corporation


                      Robot
                      Printer
                      Syringe
                      Balance
                      Bar Code Reader
                      Device Interface


                      F6=Help F7=Quit F8=Bkup F9=Up F10=Dn
```

Figure 3-1 - Teach Program Selection Menu

THE ROBOT TEACH PROGRAM

If you have only one robot in your system, the robot Teach Program options menu (see Figure 3-2) will be the first screen to appear when you choose the "Robot" Teach Program.

If you have more than one robot installed in your system, the first screen which appears when you select the "Robot" Teach Program will list your robots by name. The names come from the system configuration file, where you have assigned each module a name (e.g., robot_1, robot_artful, robot_gc). Choose the desired robot by highlighting its name and pressing Enter. The robot Teach Program options menu will then appear.

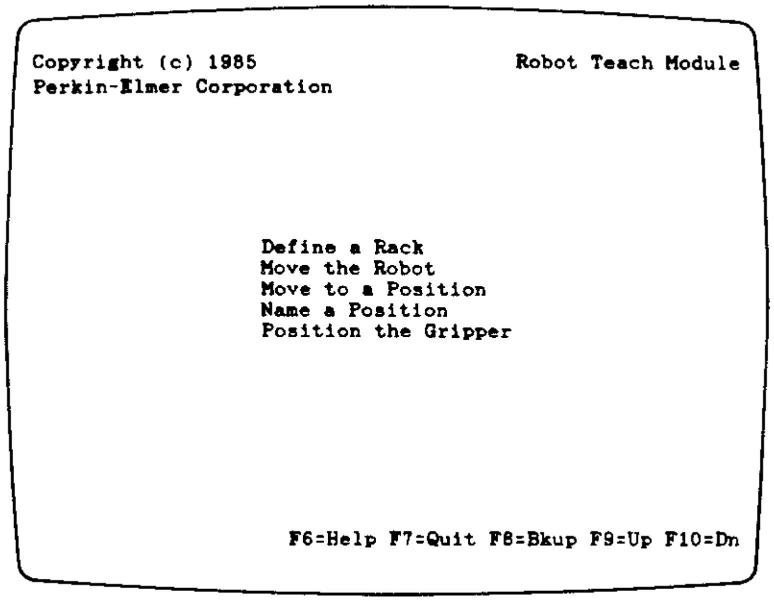
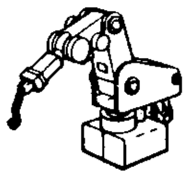

```
Copyright (c) 1985                               Robot Teach Module
Perkin-Elmer Corporation



                    Define a Rack
                    Move the Robot
                    Move to a Position
                    Name a Position
                    Position the Gripper



                      F6=Help F7=Quit F8=Bkup F9=Up F10=Dn
```

Figure 3-2 - Robot Teach Program Options

The Robot Teach Program Options Menu

Each of the procedures which the robot performs can be reduced to a series of commands. Each command specifies one of three kinds of conditions:

hand locations
gripper positions
locations in a test tube or vial rack with which the robot is to interact.



The options menu allows you to choose which of these conditions is to be specified in your new command. It lists five choices:

move the robot	- use the Teach Pendant to move the robot along the X, Y, and Z axes to the desired position.
name a position	- assign a name to the robot's present position. You may change the robot's position from within this option, before assigning a name.
move to a position	- move the robot to a previously named position by typing in the position name.
position the gripper	- open or close the grippers, name their position, and adjust the force with which they grip an object.
define a rack	- describe a test tube, pipette tip, or other rack so that the robot will be able to interact with every location in it.

The pages which follow describe each of these options in detail.

Moving the Robot

The "Move the Robot" option allows you to use the Teach Pendant to move the robot to a desired position. You can then name the position if you wish, using the "Name a Position" option. (However, if naming is your intention, it is more efficient to move the robot from within the "Name a Position" option.)

1) Highlight "Move the Robot" on the robot Teach Program options menu. Press Enter. The following messages will appear on the screen:

```
Position the Robot with the Teaching Pendant
Press the ENTER Key to Compute X, Y, Z Coordinates
```

2) Turn on the Teach Pendant, using the ON/OFF switch on the top. (See Figure 3-3.)

3) Use the six X, Y, and Z touch pad keys in the top three rows of the right column to move the robot into the desired position. Figure 3-4 indicates the direction of motion initiated by X, Y, and Z keys. Turn off the Teach Pendant.

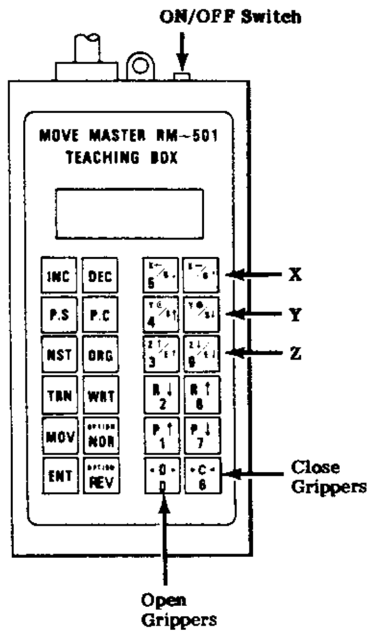


Figure 3-3 - The Teach Pendant

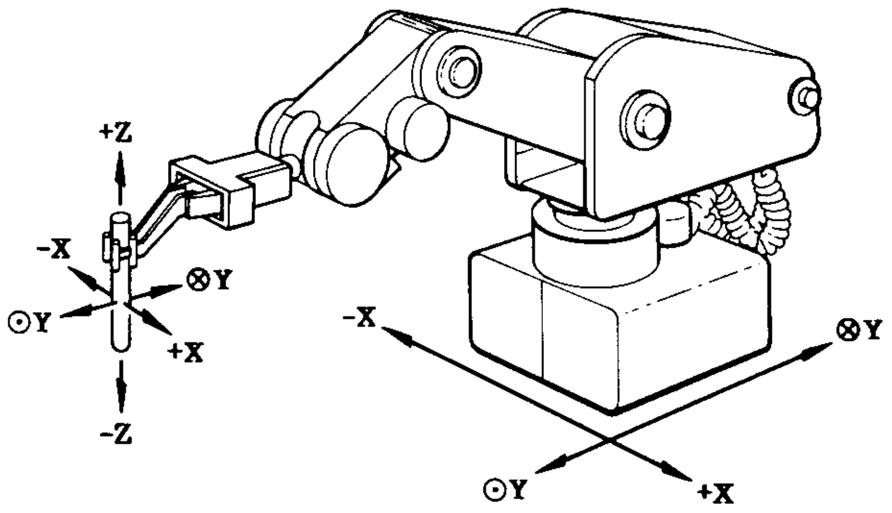
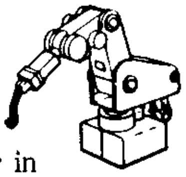


Figure 3-4 - Robot Motion in X, Y, and Z Directions



4) Press Enter. The X, Y, and Z coordinates of the position will appear in the upper right side of the screen.

5) When you have completed steps 1 - 3, you may repeat them to move to a different position. Otherwise, press F8 (Bkup) to return to the robot Teach Program options menu. If you wish to give the present position a name, choose the "Name a Position" option.

Naming a Position

The "Name a Position" option, assigns a name to the present position of the robot. When you then use that name in a PERL procedure, the robot will move to that position.

1) Highlight "Name a Position" in the robot Teach Program options menu. The screen will prompt:

   Position Name

2) Type in a name containing up to 16 characters (none of which are spaces). The first character must be alphabetic. Do not press ENTER unless you wish to assign this name to the present position of the robot.

3) If you wish to move the robot, assigning the name to a new position, turn on the Teach Pendant (see Figure 3-3). Move the robot as in the section on "Moving the Robot", above.

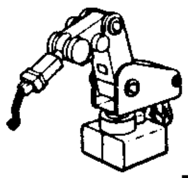
4) When the robot is in the position you wish to name, turn off the Teach Pendant. (The system will not transfer the position data with the pendant on.) Press Enter.

   The system will store the name in the PERL directory. The "Position Name" prompt will reappear.

5) Repeat steps 2 through 4 for each position you want to teach.

6) Press F8 (Bkup) when you are ready to return to the robot Teach Program options menu.

   Note: If you are teaching the robot to place a test tube in a test tube holder or the mixer, you must teach it several (at least two) intermediate positions between "over_holder" and "in_holder". Otherwise, the robot hand will describe an arc between these two extreme positions, and the test tube may break against the side of the holder. Since the "Define a Rack" option automatically teaches all the necessary intermediate positions, you can save teaching time by defining a holder or mixer as a rack with 1 column and 1 row.



Moving to a Position

If you have already assigned a name to a position, you can move the robot to it in any one of three ways:

* by typing in the position name from the PERL main menu and pressing Enter;

* by highlighting the name in the PERL Directory and pressing Enter;

* by selecting the "Move to a Position" option in the robot Teach Program.

To use the Teach Program to go to a named position, proceed as follows:

1) Highlight "Move to a Position" in the robot Teach Program options menu. The screen will prompt you:

   Position Name

2) Type in the name and press Enter. The robot will move to the named position.

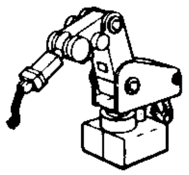
3) You may enter another position name, or use F8 (Bkup) to return to the robot Teach Program options menu.

Positioning the Gripper

When you assign a name to a robot position, the stored position includes all parameters except the position of the grippers. You must enter this in a separate step.

The gripper position has two components, the grip pressure and the width of the gripper opening. When you first highlight the "Position the Gripper" option and press Enter, a menu appears which offers the following choices:

Move the Gripper	-	open or close the grippers to the appropriate width.
Name a Gripper Position	-	assign a name to the present position of the grippers.
Name a Gripper Position/Read	-	Retrieve the value of the present gripper opening.



Set the Grip Pressure	-	set the pressure parameters (see below) if they have not previously been set appropriately for the present container being gripped (e.g., a 12 mm test tube). The default pressure is full open and full close.
Name the Grip Pressure	-	assign a name to the present grip pressure.

You will usually set the grip pressure first, if necessary, and name it. Then move the grippers to the opening width you need and name their position. Finally, you can name a command to read the present gripper opening value.

Set the Grip Pressure

1) Highlight "Set the Grip Pressure", and press Enter. You will be prompted to enter three parameters:

   Force to Close (0 - 7)
   Force to Hold (0 - 7)
   Force Duration (0 - 99)

   The Force Duration is in hundredths of a second. The Force to Close specifies an initial target current. The grippers close until that current is reached or until the duration time has elapsed; then the current changes to the holding current. The longer the time and the higher the closing current, the greater the closing force. The default settings are 4, 4, 99.

   Initially, you may need to experiment with various settings, until you find the ones which grip and hold your containers firmly, but without excessive force.

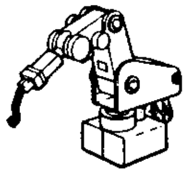
2) Type a value for each pressure parameter, pressing Enter after each one. When you have entered all three values, the gripper menu will return.

   Once the grip pressure is set for your containers, you will not need to change it again unless you change to a different size or type of container.

Name the Grip Pressure

1) Highlight "Name the Grip Pressure", and press Enter. The screen will prompt

   Grip Pressure Name:



2) Type a suitable name (up to 16 characters, none of which may be spaces), for example

    light_touch

   and press Enter. The system will store the name, and you will be returned to the gripper menu.

3) Choose another gripper option, or press F8 (Bkup) to return to the robot Teach Program options menu.

Move the Gripper

1) Highlight "Move the Gripper", and press Enter. The screen will prompt you:

```
Use Left Arrow Key to Open Grippers
Use Right Arrow Key to Close Grippers
```

2) The "hand sense" facility in your robot allows you to open and close the grippers to whatever width you need for your containers. Open or close the grippers as necessary. (See Figure 3-5.)

3) Press F8 (Bkup). You will be returned to the gripper menu.

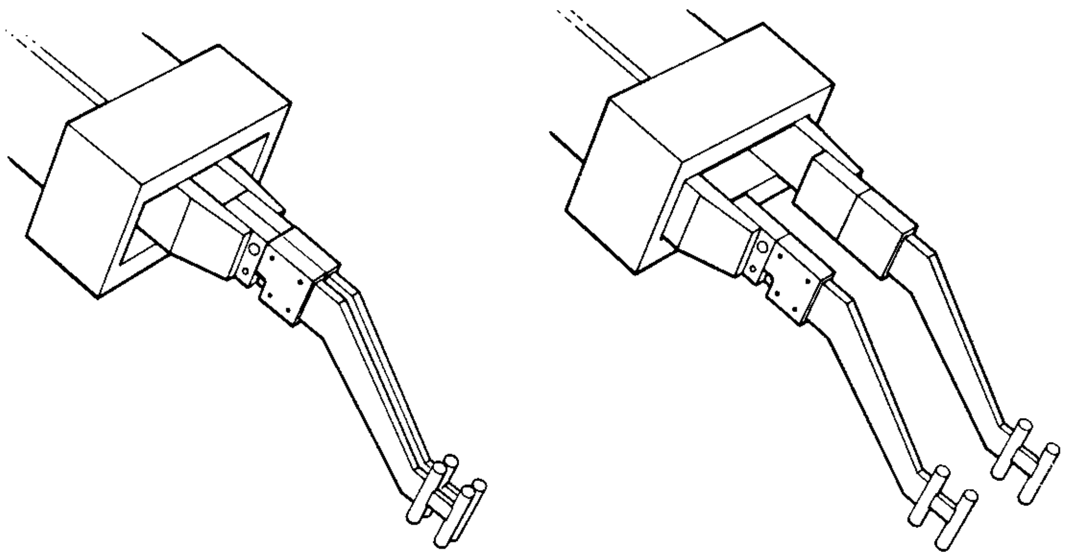
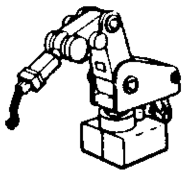


Figure 3-5 - Grippers on Robot Hand Closed and Open



Name a Gripper Position

1) Highlight "Name a Gripper Position", and press Enter. The screen will prompt you:

   Gripper Position Name

2) Type in the desired name (up to 16 characters, none of which may be spaces). Press Enter. The name will be stored, and you will be returned to the gripper menu.

3) Choose another gripper option, or press F8 (Bkup) to return to the robot Teach Program options menu.

After you have set up and named a gripper position appropriate for certain containers (for example, "soft_touch"), use that command to set the grippers at the beginning of every applications program which involves those containers.

Name a Gripper Position - Read

The "hand sense" facility assigns a value to the gripper opening size. The "Name a Gripper Position - Read" option allows you to name a command which will retrieve this value. In a PERL procedure, the value is assigned to a variable, which can be real, integer, or string. Whatever the variable type, the variable will be in a decimal representation, with values from 0 to 255.

The procedure below uses the command "hand_opn" to read the gripper position. The value of the opening is assigned to the variable x. If x is less than 10, there is no test tube in the grippers. In that case, procedure get_tube calls itself to get the next tube.

```
procedure get_tube
     sample samno%              !get tube from rack named sample
     x = hand_opn               !command to read gripper opening
     if x <= 10 then
          samno% = samno% + 1
          get_tube              !repeats procedure with next tube
     end if
end procedure
```

It is relatively unlikely that the comparison will give an exact equality, so you should use <, <=, >, >=, or ranges in such comparisons.

To name a Gripper Position Read command, proceed as follows:

1) Highlight "Name a Gripper Position - Read" on the gripper options menu.

2) Press Enter. You will be prompted to enter a command name.

3) Type in the name of the command you wish to create. The name must contain no more than 16 characters, none of which may be spaces. The first character must be alphabetic. For example,

   command name: **grip_posn**

4) Press Enter.

   The command name will be stored in the PERL Directory. The gripper options menu will return.

5) Proceed to choose another gripper option, or press F8 (Bkup) to return to the robot options menu.

Defining a Rack

The "Define a Rack" Teach Program teaches the robot to remove and replace objects which are stored in evenly spaced rows and columns.

The most obvious use for defining racks is with the MasterLab System test tube and vial racks; after the Teach Program shows the robot three of the corner locations, the robot can interact with each of the locations in the rack. However, there are many other instances in which you can use this Teach Program:

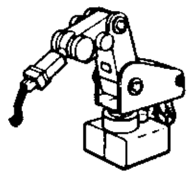
* You can use it with other kinds of racks; tilted racks and pipette tip racks, for example. Racks may be tilted at any angle, as long as the Z component of the pick-up direction is greater than zero. The wrist pitch and roll angles will be the same as when the rack was taught.

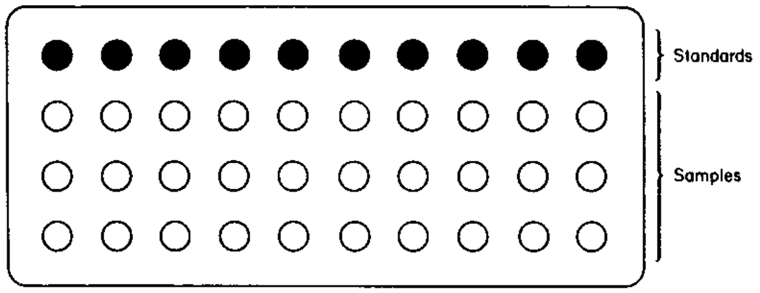
* You can treat stacked items, such as 96-well plates, as if they were a vertical rack with only one row.

* Inserting a test tube in a test tube holder (on the mixer or balance, for example) is like putting it in a rack with 1 column and 1 row.

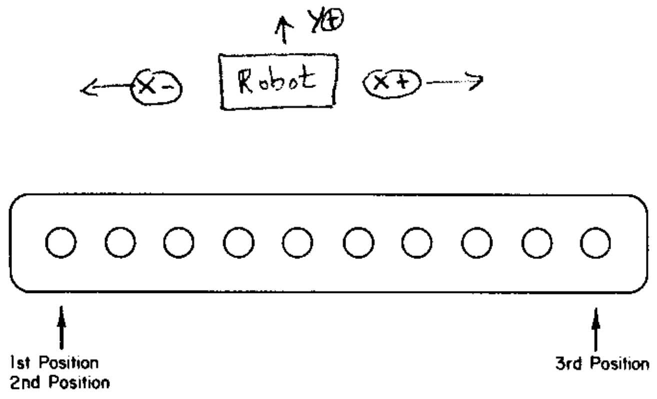
* Finally, there are racks which do not consist of orderly rows and columns. Some possibilities for handling these are described below and illustrated in Figure 3-6.

  Subdivided Racks: If you have two or more different kinds of test tubes which will be placed in the same rack (tubes of standards and tubes of samples, for example), you may subdivide the rack. For example, if the first row contained standards and the rest of the

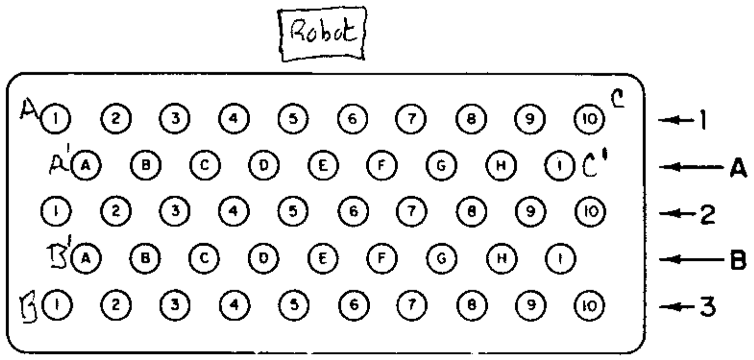




a - Subdivided Rack



b - Single-row Rack



c - Rack With Offset Rows

Figure 3-6 - Some Unusual Racks

rack contained samples, you could use the "Define a Rack" option twice, as if there were two racks. (See Figure 3-6a.) The first row could be separately defined as a rack called "standards", and the next three rows could be called "samples". (To define a one-row rack, see the following paragraph.)

Racks Consisting of a Single Row: If you wish to define a one-row rack, you must still teach the robot three positions. Teach the first position twice; it is the first location in the first row and also the first location in the last row. The third position to be taught is the last location in the row. (See Figure 3-6b.)

Racks with Offset Rows: If successive rows in a rack are offset from each other, as in Figure 3-6c, the rack must be defined as if there were two racks. The first rack consists of the numbered rows; when it is defined, location 1 in row 1 and locations 1 and 10 in row 5 are taught. The second rack consists of the lettered rows; location A in row A and locations A and I in row B are taught. The applications program will, of course, have to deal with the two racks in such a way that the samples are processed in an orderly fashion.

The following pages contain three sets of instructions for teaching racks:

* test tube racks (applicable to most other racks as well)
* pipette tip racks
* stacks of items

Test Tube Racks

1) Before you select "Define a Rack" on the options menu, you must first set the grip pressure to a low value, as follows:

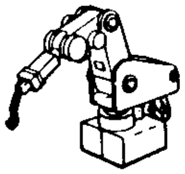
   a) Highlight "Position the Gripper" on the options menu. Press Enter.

      The gripper menu will appear.

   b) Highlight "Set the Grip Pressure" on the gripper menu. Press Enter.

      You will be prompted to enter three parameters:

      Force to Close (0 - 7)
      Force to Hold (0 - 7)
      Force Duration (0 - 99)



c) Type a value for each parameter, pressing Enter after each one. The recommended values are Force to Close 3, Force to Hold 0, Force Duration 15. Values are similar to "micromove" defined by [illegible]

> **Note:** Robot hands vary in the Force Duration needed, so you may need to try several values to find the correct one for your robot. In general, a larger value makes the hand movements coarser; a smaller value makes movements finer, but may not make the gripper open or close at all.

4, 0, 5 micromove
4, 4, 50 default
Values for [illegible] test tubes in rack definition

When you have entered all three values, the gripper menu will return.

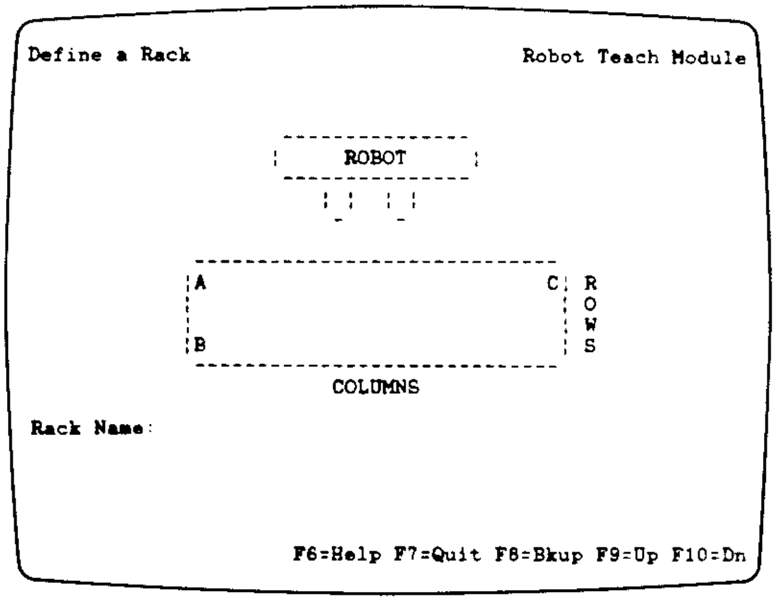
d) Press F8 (Bkup) to return to the robot options menu.

2) Highlight "Define a Rack" on the robot Teach Program options menu. Press Enter.

A prompt will ask you whether the rack to be taught is a pipette rack.

3) Type **n** to indicate this is not a pipette rack.

The screen shown in Figure 3-7 will appear, indicating the three locations (A, B, and C) you must teach the robot, and asking you for the name of the rack.



```
Define a Rack                                      Robot Teach Module

                        ----------------
                        :    ROBOT     :
                        ----------------
                          : :   : :
                          -       -

              ------------------------------
              :A                          C:  R
              :                            :  O
              :                            :  W
              :B                           :  S
              ------------------------------
                          COLUMNS

Rack Name:



                          F6=Help F7=Quit F8=Bkup F9=Up F10=Dn
```

Figure 3-7 - The "Define a Rack" Initial Screen

4) Type in a name; for example,

   Rack Name: **big_tubes**

   Press Enter.

5) The next screen contains the following instructions:

   Position the Robot to Tube Position A
   Press the ENTER Key to Compute X, Y, Z Coordinates

   To accomplish this,

   a) Turn on the Teach Pendant, using the ON/OFF switch on the top. (See Figure 3-3.)

   b) Use the six X, Y, and Z touch pad keys in the top three rows of the right column to move the robot into position above the first test tube (A in Figure 3-7). Place the hand in position so that the grippers are at the right height to grasp a test tube from the rack, near the top of the tube. (Figure 3-4 shows the direction of motion initiated by the X, Y, and Z keys.)

   Important: Teach positions A, B, and C exactly as indicated in Figure 3-7. Changing the order will cause the robot to behave unpredictably.

   c) Use the O(pen) and C(lose) touch pad keys at the bottom of the right column to adjust the grippers so that they are opened wide enough to clear the test tube, but closed enough to clear the adjacent test tube. Position A must be taught with the grip open.

   d) When the position of the robot and the position of the grippers are both correct, turn off the Teach Pendant.

   e) Press Enter. The X, Y, and Z coordinates of the position will appear in the upper right side of the screen.

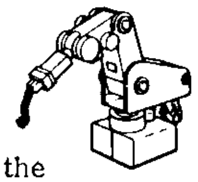
   The screen message changes to request the second tube position.

   Important: Do not change the Z (height) coordinate from this point on, unless this is a slanted rack.

6) Repeat steps a through e, above, positioning the robot above the first tube in the last row. (B in Figure 3-7.)

   The screen message then changes to request the third tube position.

7) Repeat steps a through e, positioning the robot above the last tube in the first row. (C in Figure 3-7.)



8) You will then be asked for three pieces of information, as shown in the table below. Press Enter after each response.

Prompt	Sample Response	Meaning (see Figure 3-7)
Number of rows:	**4** ENTER	Number of locations from A to B
Number of columns:	**10** ENTER	Number of locations from A to C
Length of tube (mm):	**150** ENTER	Length of containers in rack

The information about the rack named "big_tubes" will be stored in the PERL Directory. The robot Teach Program options menu returns.

9) Highlight "Position the Gripper" on the options menu. Press Enter.

10) Repeat step 1, entering the following values:

    Force to Close: **4**
    Force to Hold: **4**
    Force Duration: **50**

    } Similar to default values

11) Press the F8 (Bkup) key repeatedly, or press F7 (Quit) to return to the PERL main screen.

Pipette Tip Racks

You do not need to change the grip pressure or to adjust the gripper opening in order to teach a pipette rack.

1) Highlight "Define a Rack" on the robot Teach Program options menu.

   A prompt will ask you whether the rack to be taught is a pipette rack.

2) Type **y** (yes).

   The screen shown in Figure 3-7 will appear, indicating the three locations (A, B, and C) you must teach the robot, and asking you for the name of the rack.

3) Type in a name; for example,

   Rack Name: **tips**

   Press Enter.

4) The next screen contains the following instructions:

   Position the Robot to Tube Position A
   Press the ENTER Key to Compute X, Y, Z Coordinates

   To accomplish this,

   a) Turn on the Teach Pendant, using the ON/OFF switch on the top. (See Figure 3-3.)

   b) Use the R(oll) keys on the Teach Pendant to rotate the wrist 90 degrees, so that the pipette tip adapter is facing downwards.

   c) Use the six X, Y, and Z touch pad keys in the top three rows of the right column to move the robot into position above the first pipette tip (A in Figure 3-7). Place the hand in position so that the adapter is inserted into the tip. (Figure 3-4 shows the direction of motion initiated by the X, Y, and Z keys.)

   Important: Teach positions A, B, and C exactly as indicated in Figure 3-7. Changing the order will cause unpredictable behavior from the robot.

   d) When the position of the robot is correct, turn off the Teach Pendant.

   e) Press Enter. The X, Y, and Z coordinates of the position will appear in the upper right side of the screen.

   The screen message changes to request the second tube position.

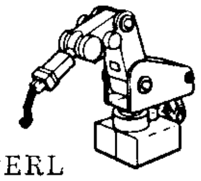
5) Repeat steps a through e, above, positioning the robot above the first tube in the last row. (B in Figure 3-7.) Keep the wrist in the proper position to pick up a pipette tip.

   The screen message then changes to request the third tube position.

6) Repeat steps a through e, positioning the robot above the last tube in the first row. (C in Figure 3-7.)

7) You will then be asked for three pieces of information, as shown in the table below. Press Enter after each response.

Prompt	Sample Response	Meaning (see Figure 3-7)
Number of rows:	**4** ENTER	Number of locations from A to B
Number of columns:	**10** ENTER	Number of locations from A to C
Length of tube (mm):	**150** ENTER	Length of pipette tips



The information about the rack named "tips" will be stored in the PERL Directory. The robot Teach Program options menu returns.

8) Press the F8 (Bkup) key repeatedly, or press F7 (Quit) to return to the PERL main screen.

Teaching Stacks of Items

A stack of items which the robot must pick up individually may be taught as a vertical rack. The "rack" has only one row, and as many columns as there are items in the stack. 96-well plates, for example, constitute such a stack.

1) Before you select "Define a Rack" on the options menu, you must first set the grip pressure to a low value, as follows:

   a) Highlight "Position the Gripper" on the options menu. Press Enter.

      The gripper menu will appear.

   b) Highlight "Set the Grip Pressure" on the gripper menu. Press Enter.

      You will be prompted to enter three parameters:

      Force to Close (0 - 7)
      Force to Hold (0 - 7)
      Force Duration (0 - 99)

   c) Type a value for each parameter, pressing Enter after each one. The recommended values are Force to Close 3, Force to Hold 0, Force Duration 15.

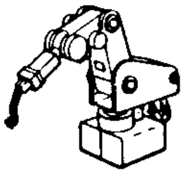
      <u>**Note:**</u> Robot hands vary in the Force Duration needed, so you may need to try several values to find the correct one for your robot. In general, a larger value makes the hand movements coarser; a smaller value makes the movements finer, but may not make the gripper open or close at all.

      When you have entered all three values, the gripper menu will return.

   d) Press F8 (Bkup) to return to the robot options menu.

2) Highlight "Define a Rack" on the robot Teach Program options menu. Press Enter.

   A prompt will ask you whether the rack to be taught is a pipette rack.



3) Type **n** to indicate this is not a pipette rack.

   The screen shown in Figure 3-7 will appear, indicating the three locations (A, B, and C) you must teach the robot, and asking you for the name of the rack.

4) Type in a name; for example,

   Rack Name: **stack2**

   Press Enter.

5) The next screen contains the following instructions:

   Position the Robot to Tube Position A
   Press the ENTER Key to Compute X, Y, Z Coordinates

   To accomplish this,

   a) Turn on the Teach Pendant, using the ON/OFF switch on the top. (See Figure 3-3.)

   b) Use the six X, Y, and Z touch pad keys in the top three rows of the right column to move the robot into position so the grippers can grasp the top item in the stack. (Figure 3-4 shows the direction of motion initiated by the X, Y, and Z keys.)

   c) Use the O(pen) and C(lose) touch pad keys at the bottom of the right column to adjust the grippers so that they are opened wide enough to clear the item, but closed enough to clear the next item below it.

   d) When the position of the robot and the position of the grippers are both correct, turn off the Teach Pendant.

   e) Press Enter. The X, Y, and Z coordinates of the position will appear in the upper right side of the screen.

      The screen message changes to request the second position.

6) Repeat steps a through e, above, positioning the robot at the bottom item in the stack.

   The screen message then changes to request the third position.

7) Repeat steps a through e again, <u>without moving the robot from the bottom item in the stack.</u>

8) You will then be asked for three pieces of information, as shown in the table below. Press Enter after each response.

Prompt	Sample Response	Meaning
Number of rows:	**1** ENTER	Stack consists of 1 vertical row
Number of columns:	**8** ENTER	Number of items in the stack
Length of tube (mm):	**50** ENTER	Height to which items are to be picked up

The information about the rack named "stack2" will be stored in the PERL Directory. The robot Teach Program options menu returns.

9) Highlight "Position the Gripper" on the options menu. Press Enter.

10) Repeat step 1, entering the following values:

Force to Close: **4**
Force to Hold: **4**
Force Duration: **50**

} Return grip to default values.

11) Press the F8 (Bkup) key repeatedly, or press F7 (Quit) to return to the PERL main screen.

THE SYRINGE TEACH PROGRAM

The syringe module has four conditions which can vary:

total syringe volume	-	The available syringes are of volumes 50, 100, 250, and 500 uL, and 1, 5, and 10 mL.
plunger position	-	the volume marking at which the plunger rests.
speed	-	May be set from 1 to 15. A setting of 1 is 1.7 seconds per full plunger stroke. Settings of from 2 to 15 are that number of seconds per full stroke.
valve position	-	set for either input or output. (See Figure 3-8.)

If your unit is the dual syringe model, the conditions can vary on each of the two syringes, except that the valves in both will always be in the same position. The two possible valve positions are illustrated in Figure 3-8.

The Teach Program for the syringe allows you to set all four parameters as you wish, then assign a name to that situation, e.g. syr_filled_5ml. Or you can use the "Name Valve Command" option to set only the valve position, without moving the plunger. You may also give a name to a syringe variable, which PERL will always interpret as referring to a syringe plunger position. You can then allow the value of that variable to increase or decrease in a program as sample is drawn in or dispensed by the syringe.

A command or variable you teach to a syringe is associated in the Directory with the module name of that syringe. This is true even if the syringes are "daisy chained", with more than one connected to a single port. Thus you do not need to use the "use" command with syringes. However, if you change the module name of your syringe, the commands may not function properly.

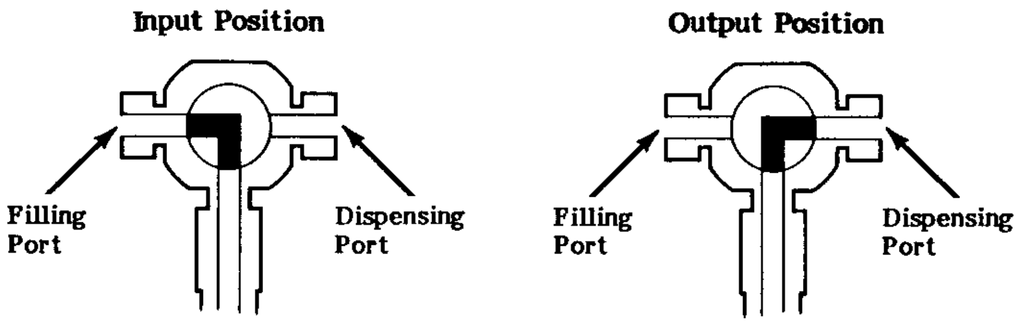


Figure 3-8 - Syringe Valve Positions, Front View

Accessing the Syringe Teach Program

1) From the PERL main menu, press the F3 (Teach) function key. The Teach Program selection menu shown in Figure 3-1 will appear. It lists the kinds of modules for which Teach Programs are available.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) keys to highlight "Syringe", and press Enter. The next screen which appears will list, by name, the syringes which are installed in your system. This information comes from the system configuration file, where you have assigned each syringe a name (e.g., syringe_1, syringe_gc).

3) Choose the desired syringe by highlighting its name and pressing Enter.

   The next menu which appears lists five options:

   Execute Syringe Command
   Name Syringe Command
   Name Syringe Variable
   Name Valve Command
   Setup Syringe Parameters

The sections which follow will describe each of these options, in the order in which they would usually be used.

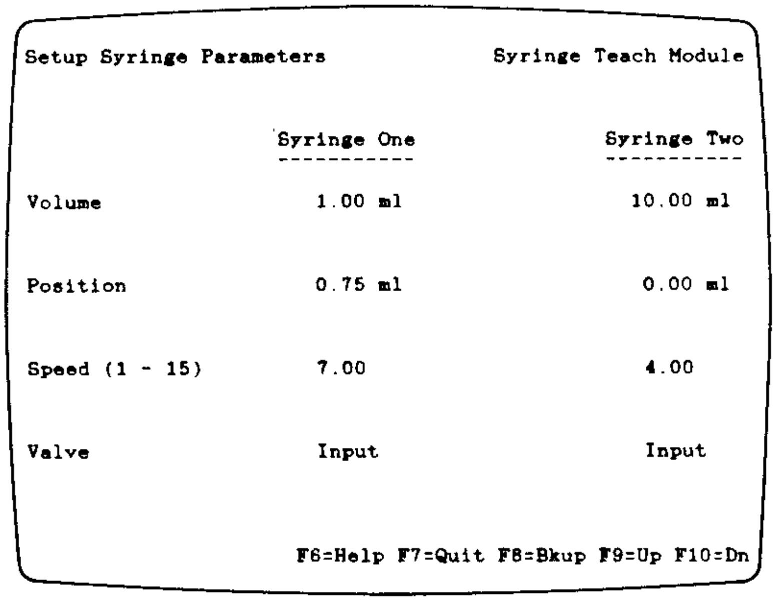
Setting Up Syringe Parameters

Before you can name or execute a syringe command, you must establish the parameters the command is to specify.

1) Highlight "Setup Syringe Parameters" on the options menu, and press Enter.

   A screen similar to the one in Figure 3-9 will appear. It lists the four syringe parameters and indicates the present value of each parameter for each of the two syringes in the module. The Position setting of 0.00 ml indicates that the syringe plunger is completely depressed.

   There is an active cursor available at the first position on the screen (Volume of Syringe One).



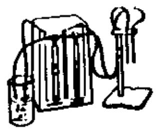
```
Setup Syringe Parameters                       Syringe Teach Module


                            Syringe One               Syringe Two
                            -----------               -----------
Volume                        1.00 ml                   10.00 ml


Position                      0.75 ml                    0.00 ml


Speed (1 - 15)                7.00                       4.00


Valve                         Input                      Input



                         F6=Help F7=Quit F8=Bkup F9=Up F10=Dn
```

Figure 3-9 - Syringe Parameter Screen

2) Use the arrow keys to move the cursor to the parameter you wish to change. Press Enter.

   The allowed values for each parameter are as follows:

   volume - 0.05 to 10.00 ml
   position - 0.00 to the volume of the syringe
   speed - 1 to 15 seconds (1 is not recommended)
   valve - input or output



3) Type in the appropriate value, such as 4.00, or o for output or i for input. Again press Enter.

4) Repeat steps 2 and 3 to make any other necessary changes.

   <u>Note:</u> The Valve parameters for the two syringes must be the same, either both input or both output. The system will change them both if either one is changed. (See Figure 3-8.) Speed settings of 2 to 15 indicate the number of seconds for a complete plunger cycle. The default value is 4.

5) When all parameters are set as you wish your new command to make them, press F8 (Bkup).

   The syringe Teach Program options menu will return. You can now proceed to name your new command; it can then be executed, to establish in the syringe module the parameter settings you have indicated.

Naming a Syringe Command

1) Highlight "Name Syringe Command" on the syringe Teach Program options menu. Press Enter.

2) When the prompt "Command Name" appears, type the name you wish to give to the arrangement of syringe parameters you established in the previous section. The name must contain no more than 16 characters, none of which may be spaces. The first character must be alphabetic. For example, you could type

   Command Name: **fill_syr_4ml**

3) Press Enter. Your command name will be stored in the PERL Directory, and the syringe Teach Program options will return to the screen.

Executing a Syringe Command

If you wish to see whether the command you constructed functions as you intended, you can use the Teach Program to have the syringe carry out the command.

1) Highlight "Execute Syringe Command" on the options menu, and press Enter. The screen will prompt "Command Name".

2) Type the name of the command you wish to execute, and press Enter.

   The syringe will execute the command. For example, if your command was "fill_syr_4ml", the plunger on the syringe should move from the 0.00 ml mark to the 4 ml mark, and 4 ml of liquid will be drawn in.

3) Proceed to enter another command, or press F8 (Bkup) to return to the syringe options menu or F7 (Quit) to return to the PERL main screen.

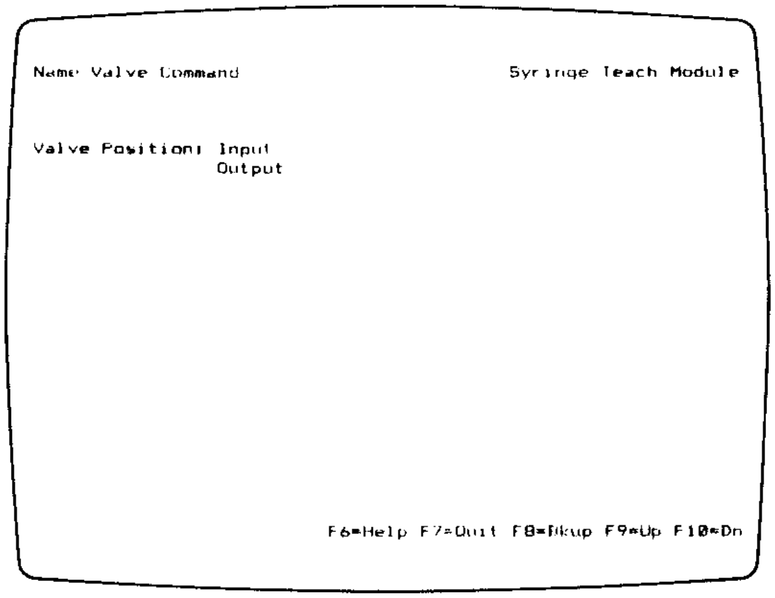
Naming a Valve Command

A syringe valve command changes the valve position from input to output, or the reverse, without changing any other syringe parameters. (For an illustration of the input and output positions, see Figure 3-8.)

1) Highlight "Name Valve Command" on the syringe Teach Program options menu. Press Enter.

   A screen similar to Figure 3-10 will appear. Highlighting will indicate whether the valve is presently set for input or output.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) keys to highlight the valve position your command is to specify.



```
Name Valve Command                                  Syringe Teach Module

Valve Position: Input
                Output

                                  F6=Help F7=Quit F8=Bkup F9=Up F10=Dn
```

Figure 3-10 - Name Valve Command Screen

3) Press the Enter key.

   The screen will prompt you for a command name.

4) Type a name which contains no more than 16 characters, none of which may be spaces. The first character must be alphabetic. For example, you could type

   Command name: **valve_in**

5) Press Enter. Your command name will be stored in the PERL Directory, and the syringe Teach Program options will return to the screen.

Naming a Syringe Variable

A syringe variable is a variable name which you store in the PERL directory by means of the syringe Teach Program. It becomes a reserved word, whose only permitted use in programs is to stand for a syringe plunger position parameter (i.e., a volume).

Suppose, for example, that you had named and stored a syringe variable called "syr_1_vol". In an applications program, you wish to fill syringe 1 with 5 ml of liquid, then dispense 0.2 ml of that liquid to each of 25 test tubes. If the sample numbers (1 to 25) are represented by i, then you must define a variable ("a", for example) such that

```
a = 5.00 - 0.2*i
```

Then your syringe variable, syr_1_vol, would be equal to a. The PERL syntax for this is

```
syr_1_vol a
```

Those two lines would be used in a program segment similar to the following:

```
fill_syr_5ml
switch_to_output
for i = 1 to 25
a = 5.00 - 0.2*i
syr_1_vol a
process_sample
next i
```

Important: The system must always execute a syringe command before it encounters a syringe variable, so that it can establish valve position, syringe volume, and syringe speed. The value of syr_1_vol could be defined by other expressions in other programs, but it will always be interpreted as referring to a syringe plunger position.

To name a syringe variable, proceed as follows:

1) Highlight "Name Syringe Variable" on the syringe Teach Program options menu. Press Enter.

2) If you have a dual syringe, the next screen will ask you whether the variable is to apply to Syringe One or Syringe Two. Highlight the one the variable is to control, and press Enter.

3) The system then prompts "Variable Name". Type in the name you wish to have reserved for use as a syringe variable.

3) Press Enter. The variable name will be stored in the PERL Directory, and the system will return you to the syringe options menu.

4) Press function key F8 (Bkup) repeatedly to return to the Teach Program selection menu and then to PERL, or press F7 (Quit) to return directly to PERL.

THE DEVICE INTERFACE TEACH PROGRAM

The device interface is used to control or interact with other devices. It can do this by turning an AC outlet on or off, by monitoring logic inputs, or by means of an output signal to the external device which turns it on or off. Thus the conditions which are given command names in the device interface Teach Program always consist of the on/off status of a particular switch or input. Some examples are shown in Table 3-2.

The names you give to the commands should be descriptive of the condition the device interface is controlling or monitoring. There could be up to twelve input lines being monitored. It is more helpful to find "instrument_ready" in the directory of available commands than to be confronted with "input_7_on".

TABLE 3-2
TYPICAL DEVICE INTERFACE COMMANDS

Condition	Possible Command Name
input from instrument ON	instrument_ready
AC outlet ON	turn_mixer_on
switch output to instrument ON	take_reading
alarm ON	sound_alarm
change has occurred in input 1	diff_at_1
change at input 2 causes AC outlet to turn ON	inpt2_turnon_mxr

Accessing the Device Interface Teach Program

1) From the PERL main menu, press the F3 (Teach) function key. The Teach Program selection menu shown in Figure 3-1 will appear. It lists the types of modules for which Teach Programs are available.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) keys to highlight "Device Interface", and press Enter. If you have more than one device interface installed in your system, the next screen which appears will list them by name. This information comes from the system configuration file, where you have assigned each module a name (e.g., dev_int, DI_1). (If you have only one device interface, this screen is omitted.)

3) Choose the desired device interface by highlighting its name and pressing Enter.

   The next menu which appears lists the four types of functions which are controlled by the device interface:

   Alarm
   Inputs
   A. C. Outlets
   Switch Outputs
   Differences
   Input Controlled Actions

   The sections which follow will describe the use of each of these options.

Using the Switch Outputs Program

The switch outputs on the device interface allow the interface to output a signal which turns an external device on or off. Such an output is used to turn on the vacuum in the crimper, for example; another is used to close the crimping jaws.

The Switch Outputs portion of the device interface Teach Program is used to create the commands which control the switch outputs, and thus in turn control the external devices to which they are connected.

When you highlight "Switch Outputs" on the device interface options menu and press Enter, the menu shown in Figure 3-11 will appear.

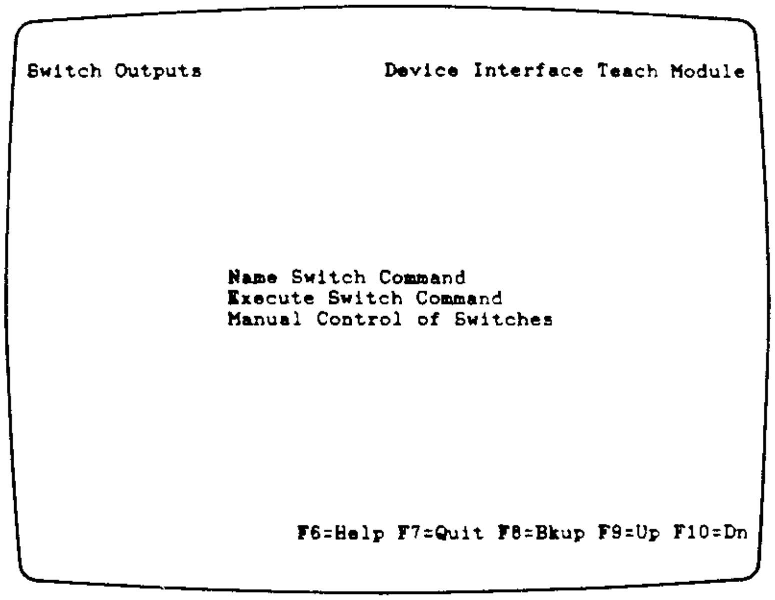

Switch Outputs
Device Interface Teach Module
Name Switch Command
Execute Switch Command
Manual Control of Switches
F6=Help F7=Quit F8=Bkup F9=Up F10=Dn

Figure 3-11 - Switch Output Menu

Manual Control of Switches

The Manual Control option is used to turn switches on and off without the use of named commands.

1) Highlight Manual Control of Switches on the Switch Outputs menu, and press Enter. A list of the twelve switches will appear. The cursor will be at switch 1.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) function keys to move the cursor to the name of the switch whose setting you wish to change. Press Enter.

3) Use the up and down arrow keys, or F9 and F10, to move the ON/OFF indicator to the setting you need.

4) Press Enter. The switch will be set as you have indicated. You may proceed to set another switch, or use the F8 (Bkup) function key to return to the Switch Outputs menu.

Name Switch Command

This option allows you to name a command which sets a switch. Every time the command is executed, the switch will go to the desired setting.

1) Highlight "Name Switch Command" on the Switch Outputs menu. Press Enter. A list of the twelve switches will appear at the left of the screen. The cursor is positioned at switch 1.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) function keys to move the cursor to the name of the switch which is to be used in the command. Press Enter.

3) Check that the ON/OFF state of the switch, as reported at the bottom of the screen, is as you wish:

   ON = closing the switch
   OFF = opening the switch

   Otherwise, change it by using the up and down arrow keys or F9 and F10. Press Enter.

4) You will be prompted to type in a command name. Type in a name having up to 16 characters, none of which may be spaces. The first character must be alphabetic. For example,

   Command Name: **crimper_on**

5) Press Enter. Your command will be stored in the PERL directory. You may proceed to move the cursor to another switch and name another command, or use the F8 (Bkup) function key to return to the Switch Outputs menu.

Execute Switch Command

After you have created a command, you can use the Execute Command option to see if the command functions as you intended.

1) Highlight "Execute Command" on the Switch Outputs menu. Press Enter. You will be prompted to enter the name of a command.

2) Type the name of the command you wish to execute, for example,

   Command Name: **crimper_off**

3) Press Enter. The command will be executed. You may proceed to enter another command to be executed, or you may use the F8 (Bkup) function key to return to the Switch Outputs menu.

Using the Inputs Program

The inputs on the device interface monitor logic inputs from external devices. One of these inputs, for example, may be connected to the vacuum device on the crimper. When the input reports that the switch to which it is connected is on, the vacuum is on. The device interface receives the input that the switch is on, but cannot change its setting. Thus there is no "Manual Control" option in the Input portion of the Teach Program.

When you highlight "Inputs" on the device interface options menu and press Enter, the menu shown in Figure 3-12 will appear.

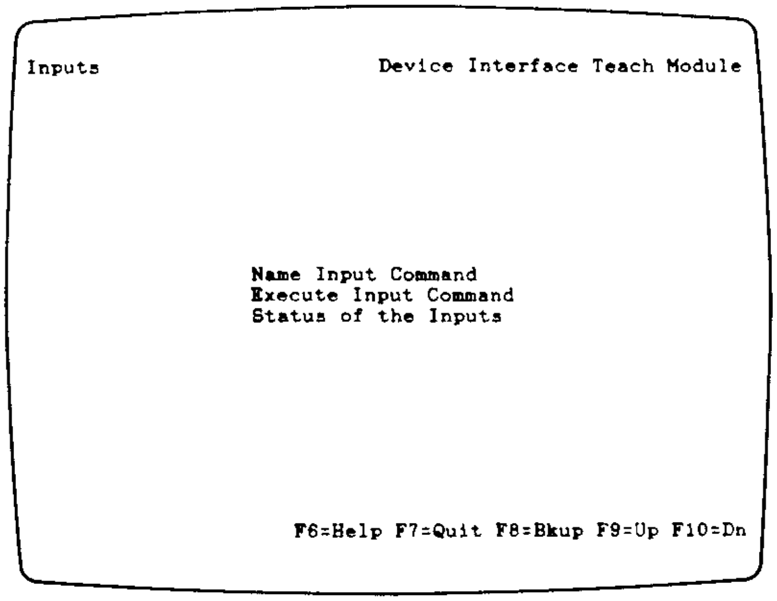

Inputs
Device Interface Teach Module
Name Input Command
Execute Input Command
Status of the Inputs
F6=Help F7=Quit F8=Bkup F9=Up F10=Dn

Figure 3-12 - Inputs Menu

Name Input Command

An input command causes the system to check the status of one of the twelve inputs. You will want to have such a command for each input to which a module is connected.

1) Highlight "Name Input Command" on the Inputs menu. Press Enter. A list of the twelve switches will appear. The cursor is positioned at switch 1.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) function keys to move the cursor to the name of the switch which is to be used in the command. Press Enter.

3) You will be prompted to type in a command name. Type in a name having up to 16 characters, none of which may be spaces. The first character must be alphabetic. For example,

   Command Name: **crimper_vac**

4) Press Enter. Your command will be stored in the PERL directory. You may proceed to move the cursor to another switch and name another command, or use the F8 (Bkup) function key to return to the Inputs menu.

   Note: The device interface Input commands are used in applications programs to monitor conditions. For example, the robot should not attempt to fill a vial whose cap was not left behind at the crimper station vacuum device. The following program segment checks that the vacuum was on:

```
x = crimper_vac
rem Variable x receives value of crimper_vac
go_to_vac
if x = 0 then
     sound_alarm
     err_rtn
end if
go_to_syringe
```

Execute Input Command

After you have created a command, you can use the Execute Input Command option to see if it functions as you intended.

1) Highlight "Execute Input Command" on the Inputs menu. Press Enter. You will be prompted to enter the name of a command.

2) Type the name of the command you wish to execute, for example,

   Command Name: **crimper_vac**

3) Press Enter. The status of the input switch used in the command will be reported. The message

   Please Press the ENTER Key to Continue

will appear. Press Enter to enter another command to be executed, or use the F8 (Bkup) function key to return to the Inputs menu, or F7 (Quit) to return to PERL.

Status of the Inputs

This option allows you to see the present ON/OFF status of each of the twelve inputs.

1) Highlight "Status of the Inputs" on the Inputs menu. Press Enter. A list of the twelve input switches will appear. Beside each the word ON appears if the input is on.

   At the bottom of the screen is the message

   Please Press the ENTER Key to Continue

2) When you are ready to return to the Input options menu, press ENTER.

Using the AC Outlets Program

The device interface has two AC outlets to which external devices can be connected. These devices are turned on and off by device interface commands created with the AC Outlets portion of the Teach Program. The mixer, for example, is such a device.

When you highlight "A. C. Outlets" on the device interface options menu and press Enter, the menu shown in Figure 3-13 will appear.

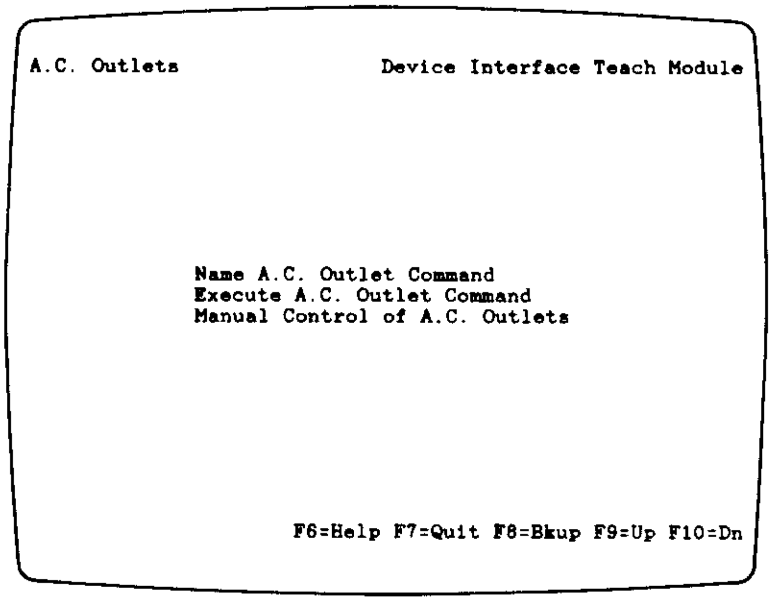
Manual Control of A.C. Outlets

You can use the Manual Control option to turn an outlet on or off without using a named command.

1) Highlight "Manual Control of A.C. Outlets" on the A.C. Outlets menu, and press Enter. A list of the outlets will appear. The cursor will be at outlet 1.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) function keys to move the cursor to the name of the outlet whose setting you wish to change. Press Enter.

3) Use the up and down arrow keys or F9 and F10 to move the ON/OFF indicator to the setting you need for your command.



```
A.C. Outlets                     Device Interface Teach Module




            Name A.C. Outlet Command
            Execute A.C. Outlet Command
            Manual Control of A.C. Outlets




                        F6=Help F7=Quit F8=Bkup F9=Up F10=Dn
```

Figure 3-13 - AC Outlets Menu

4) Press Enter. The outlet will be set as you have indicated. You may proceed to set the other outlet, or use the F8 (Bkup) function key to return to the A.C. Outlets menu.

Name A.C. Outlet Command

This option allows you to name a command which corresponds to a specified ON/OFF setting. Every time the command is executed, the switch will go to the desired setting.

1) Highlight "Name Outlet Command" on the A.C. Outlets menu. Press Enter. A list of the outlets will appear at the right of the screen. The cursor is positioned at A.C. outlet 1.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) function keys to move the cursor to the name of the outlet which is to be used in the command. Press Enter.

   Check that the ON/OFF state of the switch, as reported at the bottom of the screen, is as you wish. Otherwise, change it by using the up and down arrow keys or F9 and F10. Press Enter.

3) You will be prompted to type in a command name. Type in a name having up to 16 characters, none of which may be a space. The first character must be alphabetic. For example,

   Command Name: **mixer_on**

4) Press Enter. Your command will be stored in the PERL directory. You may proceed to move the cursor to the other outlet and name another command, or use the F8 (Bkup) function key to return to the A.C. Outlets menu.

Execute A.C. Outlet Command

After you have created a command, you can use the Execute Command option to see if the command functions as you intended.

1) Highlight "Execute Command" on the A.C. Outlets menu. Press Enter. You will be prompted to enter the name of a command.

2) Type the name of the command you wish to execute, for example,

   Command Name: **mixer_off**

3) Press Enter. The command will be executed. You may proceed to enter another command to be executed, or you may use the F8 (Bkup) function key to return to the A.C. Outlets menu.

Using the Alarm Program

There is an alarm permanently installed in the device interface. Applications programs can command that alarm to sound when a certain condition occurs:

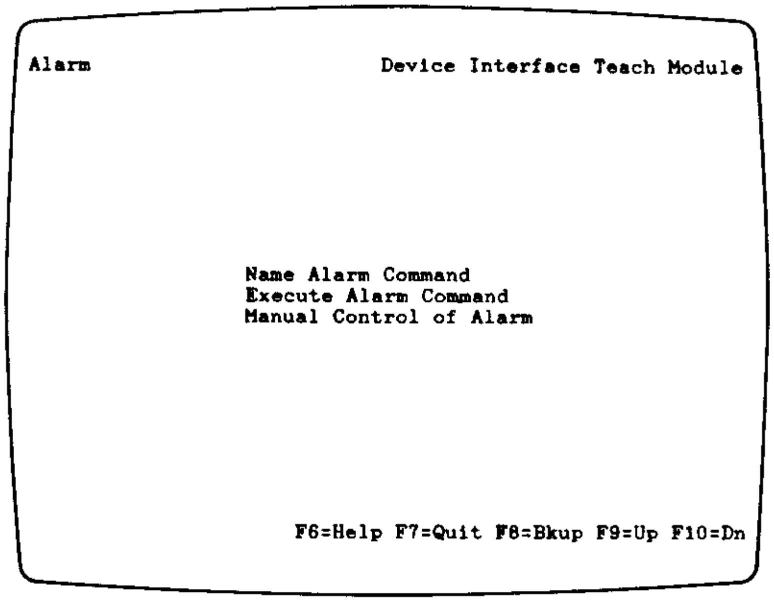
```
x = inst_not_ready
if x = 1 then
     sound_alarm
end if
```

When you highlight "Alarm" on the device interface options menu and press Enter, the menu shown in Figure 3-14 will appear.

Manual Control of Alarm

The Manual Control option can be used to set an alarm without using a named command.

1) Highlight "Manual Control of Alarm" on the Alarm menu, and press Enter. The present ON/OFF status of the alarm will be given.


Alarm Device Interface Teach Module

Name Alarm Command
Execute Alarm Command
Manual Control of Alarm

F6=Help F7=Quit F8=Bkup F9=Up F10=Dn

Figure 3-14 - Alarm Menu

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) function keys to move the ON/OFF indicator to the setting you need for your command.

3) Press Enter. The alarm will be set as you have indicated. Use the F8 (Bkup) function key to return to the Alarm menu.

Name Alarm Command

This option allows you to name a command which corresponds to a specified ON/OFF alarm setting. Every time the command is executed, the alarm will be turned on or off as specified.

1) Highlight "Name Alarm Command" on the Alarm menu. Press Enter.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) function keys to move the cursor to the desired setting. Press Enter.

3) You will be prompted to type in a command name. Type in a name having up to 16 characters, none of which may be a space. The first character must be alphabetic. For example,

   Command Name: **sound_alarm**

4) Press Enter. Your command will be stored in the PERL directory. Use the F8 (Bkup) function key to return to the Alarm menu.

Execute Alarm Command

After you have created a command, you can use the Execute Command option to see if the command functions as you intended.

1) Highlight "Execute Command" on the Alarm menu. Press Enter. You will be prompted to enter the name of a command.

2) Type the name of the command you wish to execute, for example,

   Command Name: **alarm_off**

3) Press Enter. The command will be executed. You may proceed to enter another command to be executed, or you may use the F8 (Bkup) function key to return to the Alarm menu.

Using the Differences Program

A difference command detects a transient change in one of the inputs to the device interface. It returns a value of 0 if no change has occurred; a value of 1 indicates a change.

A PERL procedure must execute the difference command once to arm it, before actually looking for a change. Once the command is armed, it can be used for as long as the device interface continues to be powered.

When you highlight "Differences" on the device interface options menu and press Enter, the menu shown in Figure 3-15 will appear.

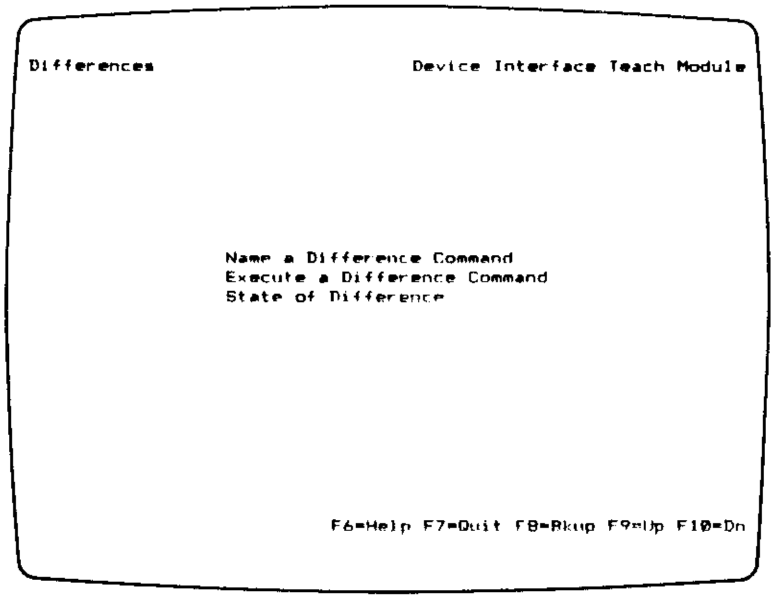

Differences
Device Interface Teach Module
Name a Difference Command
Execute a Difference Command
State of Difference
F6=Help F7=Quit F8=Bkup F9=Up F10=Dn

Figure 3-15 - Difference Menu

Name Difference Command

An difference command causes the system to check the status of one of the twelve inputs. You could have such a command for each input to which a module is connected.

1) Highlight "Name Difference Command" on the Differences menu. Press Enter. A list of the twelve inputs will appear. The cursor is positioned at input 1.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) function keys to move the cursor to the name of the switch which is to be used in the command. Press Enter.

3) You will be prompted to type in a command name. Type in a name having up to 16 characters, none of which may be spaces. The first character must be alphabetic. For example,

   Command Name: **diff_1**

4) Press Enter. Your command will be stored in the PERL directory. You may proceed to move the cursor to another switch and name another command, or use the F8 (Bkup) function key to return to the Differences menu.

   Note: The device interface Difference commands are used in applications procedures to monitor conditions. For example, the following procedure segment arms the "diff_1" command, assigning its value (0, when first armed) to the variable x. Later, the procedure uses the command again. If a change has occurred, so that the value is 1, an error routine will be executed.

```
x = diff_1
rem Variable x receives value of diff_1
while x = 0
     x = diff_1
     get_samp
     process
end while
if x = 1 then
     sound_alarm
     err_rtn
end if
```

Execute Difference Command

After you have created a command, you can use the Execute Difference Command option to see if it functions as you intended.

1) Highlight "Execute Difference Command" on the Differences menu. Press Enter. You will be prompted to enter the name of a command.

2) Type the name of the command you wish to execute, for example,

    Command Name: **diff_4**

3) Press Enter. The state of the input used in the command will be reported. It will be either 0 (if no change has occurred) or 1 (if there has been a change). The message

    Please Press the ENTER Key to Continue

    will appear.

4) Press Enter to enter another command to be executed, or use the F8 (Bkup) function key to return to the Differences menu, or F7 (Quit) to return to PERL.

State of the Differences

This option allows you to see the present ON/OFF status of each of the twelve inputs.

1) Highlight "Status of the Differences" on the Differences menu. Press Enter.

    A list of the twelve input switches will appear. Beside each will appear either a 0 (if no change has occurred) or a 1 (if there has been a change).

    At the bottom of the screen is the message

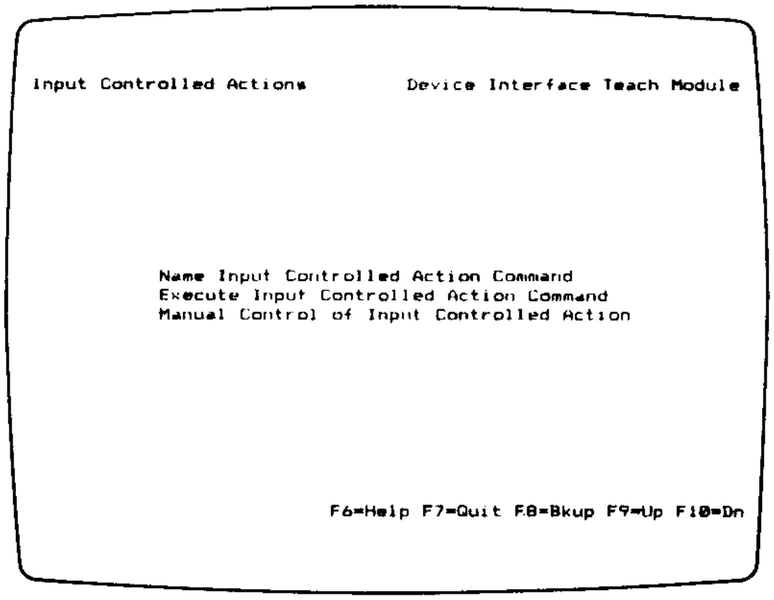
    Please Press the ENTER Key to Continue

2) When you are ready to return to the Difference options menu, press ENTER.

Using the Input Controlled Actions Program

Input controlled action commands allow the device interface to autonomously control an output, in response to a change in an input. This is useful whenever the response to a changed input needs to be immediate. You could, for example, have the device interface turn on the output to an instrument ("take_reading") as soon as the input from that instrument indicates the instrument is ready.

When you highlight "Input Controlled Actions" on the device interface options menu and press Enter, the menu shown in Figure 3-16 will appear.



```
Input Controlled Actions          Device Interface Teach Module



          Name Input Controlled Action Command
          Execute Input Controlled Action Command
          Manual Control of Input Controlled Action



                         F6=Help F7=Quit F8=Bkup F9=Up F10=Dn
```

Figure 3-16 - Input Controlled Actions Menu

Manual Control of Input Controlled Action

The Manual Control option is used to turn switches on and off without the use of named commands.

1) Highlight Manual Control of Input Controlled Action on the Input Controlled Actions menu, and press Enter. A list of the two AC outlets, the alarm, and the twelve switch outputs will appear. The cursor will be at AC outlet 1.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) function keys to move the cursor to the name of the switch whose setting you wish to change. Press Enter.

3) Use the up and down arrow keys, or F9 and F10, to move the ON/OFF indicator (for AC outlets and alarm) or the OPEN/CLOSE indicator (for switch outputs) to the setting you need.

4) Press Enter. The switch will be set as you have indicated. You may proceed to set another switch, or use the F8 (Bkup) function key to return to the Input Controlled Actions menu.

Name Input Controlled Action Command

This option allows you to name a command which sets a switch in response to a specified condition of an input. You must select five conditions before you can name the command:

1. the particular input whose condition will control the action;
2. the input state which will trigger the action (whether the value is to be high or low);
3. the switch which is to be acted upon (an AC outlet, the alarm, or an output switch);
4. whether the switch is to be turned ON or OFF.
5. whether PERL is to wait for the action to be completed before continuing with the rest of the procedure. For example, after loading a sample into an LC injector, the robot can go on to prepare the next sample, without waiting; when the instrument is ready, the sample is injected. On the other hand, if the action command is to turn off a shaker table, the robot will have to wait until the table has been turned off before it can access the table.

Important: If the system executes any other device interface command while waiting for the input which triggers the input controlled action command, the latter will be overwritten and will never be executed.

1) Highlight "Name Input Controlled Action Command" on the Switch Outputs menu. Press Enter. A list of the twelve inputs will appear at the left of the screen. The cursor is positioned at the first.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) function keys to move the cursor to the name of the input which is to be used in the command. Press Enter.

3) At the bottom left of the screen, highlight the state of the input which will trigger the action to occur, HIGH or LOW. Press Enter.

   A list of the available outputs will appear at the right of the screen.

4) Highlight the output (either AC outlet, alarm, or switch output) that you want the command to affect. Press Enter.

5) Change the highlighting on the ON/OFF indicator (for AC outlets and alarm) or the OPEN/CLOSE indicator (for switch outputs) to the setting you want the command to establish.

6) The words "Wait" and "No Wait" appear at the bottom of the screen. Use the right and left arrow keys to change the highlighting to indicate whether you want PERL to pause and wait for the action to be carried out.

7) You will be prompted to type in a command name. Type in a name having up to 16 characters, none of which may be spaces. The first character must be alphabetic. For example,

   Command Name: **inst_on**

8) Press Enter. Your command will be stored in the PERL directory. You may proceed to name another command, or use the F8 (Bkup) function key to return to the Input Controlled Actions menu.

Execute Input Controlled Action Command

After you have created a command, you can use the Execute Command option to see if the command functions as you intended.

1) Highlight "Execute Input Controlled Action" on the Input Controlled Actions menu. Press Enter.

   You will be prompted to enter the name of a command.

2) Type the name of the command you wish to execute, for example,

   Command Name: **inst_off**

3) Press Enter. The command will be executed.

   You may proceed to enter another command to be executed, or you may use the F8 (Bkup) function key to return to the Input Controlled Actions menu.

THE BALANCE TEACH PROGRAM

The output of a balance is always a measured weight. A "weigh command" is a command to the System Controller to accept this reading, so it can be used as data. The commands

```
get_sample_wt
weigh_test_tube
record_vial_wt
```

all cause this same action to be performed, but they have been given different names for increased program clarity.

Before a procedure can use the measured mass, it must assign the mass to a variable. For example, the following procedure weighs a sample, transfers the measured mass to a variable called "mass", then appends mass to a file on disk.

```
procedure store_wt
mass = get_weight
append mass to "mass_file"
close "mass_file"
end procedure
```

Accessing the Balance Teach Program

1) From the PERL main menu, press the F3 (Teach) function key. The Teach Program selection menu shown in Figure 3-1 will appear. It lists the types of modules for which Teach Programs are available.

2) Use the up and down arrow keys or the F9 (Up) and F10 (Dn) keys to highlight "Balance" and press Enter. If you have more than one balance installed in your system, the next screen which appears will list them by name. This information comes from the system configuration file, where you have assigned each module a name (e.g., weigh_station, balance_1). (If you have only one balance, this screen is omitted.)

3) Choose the desired balance by highlighting its name and pressing Enter.

   The next menu which appears lists three options:

   ```
   Display Current Weight
   Name Weigh Command
   Execute Weigh Command
   ```

   The sections which follow will describe each of these options.

   Note: If you have a Sartorius balance, the menu will also include "Name Tare Command".

Displaying the Current Weight

1) Highlight "Display Current Weight" on the balance options menu.

2) Press Enter. The current weight being output by the balance is displayed on the screen. You may select another option, or press F8 (Bkup) to return to the Teach Program selection menu.

Naming a Weigh Command

1) Highlight "Name Weigh Command" on the balance options menu.

2) Press Enter. You will be prompted to enter a command name.

3) Type in the name of the command you wish to create. The name must contain no more than 16 characters, none of which may be spaces. The first character must be alphabetic. For example,

   command name: **get_weight**

4) Press Enter. The command name will be stored in the PERL Directory.

5) Proceed to name another command, or press F8 (Bkup) to return to the balance options menu.

Executing a Weigh Command

After you have created a command, you can use the Execute Weigh Command option to see if the command functions as you intended.

1) Highlight "Execute Weigh Command" on the balance options menu.

2) Press Enter. You will be prompted to enter a command name.

3) Type in the name of the command you wish to have executed.

4) Press Enter. The command will be executed, and the present weight will be displayed.

5) Proceed to type in the name of another command to be executed, or press F8 (Bkup) to return to the balance options menu.

Naming a Tare Command (Sartorius Balances)

Because the Sartorius MP-8 is a bidirectional interface, you can use a command from PERL to tare the balance. When you use a tare command in a procedure, follow it with a weigh command, to ensure that the tare command was properly executed.

1) Highlight "Name Tare Command" on the balance options menu.

2) Press Enter. You will be prompted to enter a command name.

3) Type in the name of the command you wish to create. The name must contain no more than 16 characters, none of which may be spaces. The first character must be alphabetic. For example,

   command name: **tare_it**

4) Press Enter. The command name will be stored in the PERL Directory.

5) Proceed to name another command, or press F8 (Bkup) to return to the balance options menu.

4 THE PERL LANGUAGE

4 THE PERL LANGUAGE

ELEMENTS OF PERL

PERL Statements

A program in PERL, called a "procedure", consists of a series of one-line statements. As with any program, the statements constitute a set of commands which the system follows in order. The following are typical PERL statements:

```
use robot_2
set timer_1 for 10 seconds
get_sample_tube
display "All samples complete."
```

The statements are of four general types:

- Statements with syntax similar to the BASIC language which control the flow of the program. Such statements as "if...then", "input", or "for...next" will be familiar if you know BASIC. If you do not, you will find that the simple English words used will make the meaning easy to grasp, although you may wish to consult a BASIC textbook. (A good introductory text is BASIC, a Self-Teaching Guide, by Robert Albrecht, Leroy Finkel, and Jerald Brown, published by John Wiley and Sons.)

- Statements used for input and output of information to various devices (e.g., monitor, printer, or disk drive). These statements are also English words, and in many cases are similar to commands you may have encountered in using other systems. Examples are "display", "read", and "print".

- Statements unique to PERL which are needed to control a robot carrying out laboratory procedures. Such statements as "set timer" and "use" are typical.
- Statements which have been created through the use of the Teach Programs. Typical examples would be "get_sample_tube", or "move_to_syringe". These statements may have up to 16 characters, none of which may be spaces. (The latter restriction accounts for the frequent presence of the underline character.)

This section will be concerned primarily with the first three types of statement. The creation of operator-defined statements is described in Section 3, The Teach Programs.

Arguments Used in PERL Statements

With the exception of the operator-defined statements created with the Teach Programs and unique to each laboratory ("over_the_mixer", for example), most statements in PERL require some additional information, called arguments, after the word which begins the statement. In the statement formats

```
use (device name)
for (variable) = (expression 1) to (expression 2) [step (expression 3)]
print (desired output)
```

the parentheses specify arguments required in the "use", "for", and "print" statements. The arguments may be either constants, variables, expressions, or functions of any of these.

Constants and Variables

PERL uses either numeric or character constants. The numeric constants may be either real, having a decimal point and possibly a decimal part, or they may be integer, having no decimal point.

Variables, to which the system assigns values during procedure execution, are of the same types as constants. Variables have names which may contain up to 16 characters, none of which may be spaces. Character variable names end with a dollar sign; integer variable names end with a per cent sign. If you do not specify character or integer data type, PERL assumes that the variable is of real data type. Table 4-1 gives examples of the types of variables and constants PERL supports.

TABLE 4-1
DATA TYPES IN PERL

Type	Examples of Constants	Examples of Variables
Numeric		
Real	2.4, 40., 10.0	ml_of_standard
Integer	3, 74	num_of_samples%
Character	"newstuff"	prodname$

Array Variables

Variables of the same type may be grouped together and assigned the same variable name. Such a grouping is called an array. The variables in the array are distinguished from each other by a subscript in parentheses following the name. For example, the weights of 10 samples could be placed in an array called sampwt. The individual weights would be sampwt(1), sampwt(2), etc. A "dim" (dimension) statement must always precede the first use of an array variable in a procedure. See the description of the "for...next" statement for PERL statements which involve arrays.

Expressions

Constants and variables may be joined by arithmetic operators (see Table 4-2) or relational operators (see Table 4-3) to form expressions. Such expressions may be used in PERL statements in the same way that constants or variables are used. The following statements use expressions as arguments:

```
for i = 1 to (num_of_samples/2)

result = absorbance/sample_weight
```

TABLE 4-2
ARITHMETIC OPERATORS

Operation	Operator	Expression
Addition	+	a + b
Subtraction	-	a - b
Multiplication	*	a*b
Division	/	a/b
Modulus	@	a @ b
Negation	-	-a

TABLE 4-3
RELATIONAL AND LOGICAL OPERATORS

Operator	Operation
=	equal to
<>	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
AND	logical AND
OR	logical OR
NOT	logical NOT

When PERL evaluates an expression such as 2+4*3, it operates under a hierarchy that determines which operation should come first. Without such a hierarchy, the expression could be evaluated as either 14 or 18. (It is 14.)

The following list shows the order of precedence. The system performs the operations at the top of the list first, from left to right in the expression, followed in order by the items lower on the list.

1. Functions.
2. Unary subtraction; NOT.
3. Multiplication; division; modulus
4. Addition and subtraction.
5. <=; >=; <; >
6. =; <>
7. Exponentiation.
8. AND; OR

Functions

Certain numeric and character functions are preprogrammed in PERL. You can use them simply by naming the function and the variable on which it is to be used. For example, the absolute value function is named "abs". The following statement would cause the absolute value of the expression (3*a-5) to be displayed on the monitor:

```
display abs(3*a-5)
```

If a were equal to -4, the result displayed would be 17.

The functions supported by PERL are listed in Tables 4-4 and 4-5, with their formats and an example of each.

TABLE 4-4 NUMERIC FUNCTIONS

Important: All angles are expressed in radians.

Function	Format	Example
Absolute Value	abs(x)	display abs(25-73) 48
Arccosine	arccos(x)	display arccos(0.9659262) 0.261798
Arcsine	arcsin(x)	display arcsin(0.2588177) 0.261798
Arctangent	arctan(x)	display arctan(3) 1.249046
Cosine	cos(x)	display cos(0.2617980) 0.965928
Cosecant	csc(x)	
Cotangent	cot(x)	display cot(1.047192) 0.577358
Exponent (returns e to the xth power)	exp(x)	display exp(1) 2.718282
Exponent to base 10 (returns 10 to the xth power)	exp10(x)	display exp10(2) 100
Integer (Note that the number is truncated. To round, use int(x + 0.5).	int(x)	display int(123.567) 123
Log (to base e)	log(x)	display log(2) 0.693147
Log (to base 10)	log10(x)	display log10(1000) 3
Sine	sin(x)	display sin(0.2617980) 0.258818
Square Root	sqrt(x)	display sqrt(2) 1.414214
Tangent	tan(x)	display tan(1.047192) 1.732029

TABLE 4-5 STRING FUNCTIONS

Function	Format	Example

Concatenation Joins character strings together	+	a$ = "Master"
		b$ = "Lab System"
		display a$ + b$
		    MasterLab System

ASCII String Function (returns the ASCII code of the character)	asc(x$)	display asc("a") 65
Character function (returns the character whose decimal ASCII code is given)	chr$(x)	display chr$(65) A

Inkey (pauses until a character is input from the keyboard, then accepts it)	x$ = inkey$	display "continue? - y or n"
		x$ = inkey$
		if x$ = "y" then
		    process
		else
		    system
		end if

Instring (returns the position of the 1st occurrence of y$ in x$. May begin searching at the nth position.)	instr(x$,y$)
	instr(n,x$,y$)
		x$ = "abcdeb"
		y$ = "b"
		name$ = instr(x$,y$)
		newname$ = instr(4,x$,y$)
		display name$
		display newname$
		    2
		    6

Left characters (returns the left n characters)	left$(x$,n)	x$ = "Thanksgiving"
		display left$(x$,6)
		    Thanks

Length (returns the no. of characters, including spaces, in x$)	len(x$)	x$ = "Greenville, ME"
		display len(x$)
		    14

Middle characters (returns the characters from i to the end)	mid$(x$,i)
	mid$(x$,i,n)
		x$ = "smiles"
		display mid$(x$,2)
		    miles

Function	Format	Example
Readkey$ (accepts a	x$ = readkey$	display "Press any key to stop"
character input	if readkey$ then	for i = 1 to 100
from keyboard, but		x$ = readkey$
does not pause if		if x$ <> "" then
none is input)		break
		else
		process
		end if
		next i
Right characters	right$(x$,n)	x$ = "brink"
(returns the		display right$(x$,3)
right n charac-		ink
ters		
String (returns	str$(x)	display str$(365)
a character		365
constant which		
corresponds to		
a numeric one)		
Value (Gives the	val(x$)	temp$ = "25 C"
numeric value of		display val(temp$)
a character		25.000
string)		

PERL STATEMENTS

This section describes the statements available in PERL, grouped according to function. The following alphabetical listing gives the page number of each description.

You will find a brief summary of the format of each statement and a definition of its use in the Reference Guide to PERL at the back of this manual.

append (var) to (filename)	(page 4-40)
as	(page 4-39)
break	(page 4-37)
case (var) or case (expr)	(page 4-32)
clear	(page 4-24)
close	(page 4-18)
close (filename)	(page 4-40)
continuous	(page 4-19)
cursor (h line no.),(v column no.)	(page 4-24)
date$	(page 4-43)
day$	(page 4-43)
default	(page 4-32)
define (var) as (port name)	(page 4-39)
devices	(page 4-17)
dim (var(i))	(page 4-36)
display (desired output)	(page 4-24)
display devices	(page 4-17)
dos	(page 4-13)
down (distance)	(page 4-18)

else	(page 4-31)
end procedure	(page 4-11)
eof	(page 4-42)
for (var) = (expression) to (expression) [step (expression 3)]	(page 4-34)
from	(page 4-28 and 4-40)
if (expression) then	(page 4-31)
init (device)	(page 4-17)
inkey$	(page 4-28)
input [prompt] (variable)	(page 4-25)
is	(page 4-32)
library	(page 4-15)
load	(page 4-14)
log (destination)	(page 4-16)
month$	(page 4-43)
next (var)	(page 4-34)
open	(page 4-18)
output	(page 4-28)
perl	(page 4-13)
print	(page 4-30)
print screen	(page 4-30)
procedure (name)	(page 4-11)
read (var) from (filename)	(page 4-40)
readkey$	(page 4-29)
receive (var)	(page 4-39)
redirect input from (device), or redirect output to (device)	(page 4-28)

relative (x,y,z,p,r)	(page 4-18)
rem (comment)	(page 4-11)
rewind (filename)	(page 4-40)
save	(page 4-14)
screen	(page 4-30)
send (var)	(page 4-39)
set timer (no.) for (quantity of time) (type of time)	(page 4-20)
speed (integer)	(page 4-19)
step	(page 4-34)
stop	(page 4-14)
suspend	(page 4-14)
system	(page 4-14)
then	(page 4-31)
time$	(page 4-43)
timer	(page 4-20)
to	(page 4-34)
until (condition)	(page 4-22)
up (distance)	(page 4-18)
use (device name)	(page 4-17)
wait for timer (no.)	(page 4-20)
while (condition)	(page 4-22)
write (var) to (filename)	(page 4-40)
!	(page 4-12)
\	(page 4-26)

Housekeeping Statements

procedure (name)

end procedure

Every PERL applications program, and every subroutine within it, is a procedure which the system is to follow. Each such procedure begins with a "procedure" statement which names it:

```
procedure uv
procedure weigh
procedure mix
```

You can use the procedure name as a line in other procedures, telling the system to temporarily branch to the named procedure, follow it as a subroutine, then return to the original procedure. For example, the line

```
weigh
```

would cause the system to execute the procedure which begins "procedure weigh".

The "end procedure" statement signals the last line in a procedure or subroutine. No procedure name need be given; the system assumes that the statement refers to the last previous "procedure" statement.

rem

It is good programming practice to insert in the procedure comment lines which indicate what the procedure is supposed to be doing at that point. These are called remarks or "rem" statements. For example,

```
rem   subroutine positions sample tube above mixer
rem   if error detected, prints message
rem   v3 is the concentration of HCl
```

The system does not attempt to take any action on a line which begins with the word "rem". However, the line remains with the procedure and is printed for the programmer's information whenever the procedure is listed.

You can use "rem" only at the beginning of a statement. If you wish to append a remark to the end of a statement line, begin it with an exclamation mark (!):

```
a = 5.00 - 0.2*i  !  volume remaining in syringe
```

You can also use the exclamation mark at the beginning of a statement, in place of "rem".

System Control

```
dos
dos [command]
dos [filename]
perl
```

The "dos" command, without an argument, halts execution of a running procedure and returns control to the operating system (DOS). You can then issue DOS commands from the keyboard. PERL remains in memory; when you type "perl", the procedure resumes execution.

However, you will usually want to use either a valid DOS command or the name of a batch file as the argument to "dos". The system will execute the command or the tasks listed in the batch file, then automatically resume execution of the PERL procedure.

For example, the following procedure uses the "dos" command to execute a BASIC program called "acquire":

```
procedure command
     deliver                 ! deliver sample
     dos basic acquire       ! call basic program acquire.bas
                             ! to acquire data from instrument
     cleanup                 ! prepare for the next sample
end procedure
```

The procedure listed below executes the tasks in a batch file called "data.bat":

```
procedure batch
     deliver
     dos data                ! call data.bat to acquire data, reduce
                             ! data, copy data to new directory
     cleanup
end procedure

listing of data.bat:
     basic acquire
     reduce
     copy newdata \data\july15.dat
```

system

The "dos" and "system" commands both return control to the operating system (DOS). They differ in that "system" aborts PERL, while "dos" leaves PERL in memory.

stop

suspend

If you put "suspend" in your procedure, execution will halt at that point as if you had pressed F7 (Halt). You can then continue execution by pressing F8 (continue), or you can abort the program by pressing F7.

A "stop" statement in the procedure aborts execution and returns you to the PERL Direct Command Processor. It is equivalent to pressing F7 twice in succession.

load

save

The PERL Directory stores in the file PERL.DIR all commands you create with the Teach Programs. The "load" command tells the system to read PERL.DIR into memory. Any new Directory entries presently in memory will be lost. The system automatically makes a backup file (PERLDIR.BAK) of the Directory being loaded.

To write the Directory presently in memory to disk, use "save". The system will write the Directory to PERL.DIR.

library

The "library" command gives you a way to shorten the run-time devoted to waiting for subprocedures to load as they are needed. Use it as described below.

1) Use the Editor to create a file which contains <u>all</u> the procedures you will need in your overall procedure. Do this by loading them into the Editor in sequence. The system will use the name of the first procedure you load as the library name.

```
procedure fill_syr
      .
      .
      .
end procedure
procedure disp_liq
      .
      .
      .
end procedure
procedure mix_samp
      .
      .
etc.
```

2) Create a procedure for loading the file created in (1), above. Give it a name (for example, "gc_libr"). It will use the "library" command and be similar to the following:

```
procedure gc_libr
library fill_syr
end procedure
```

3) In your main PERL procedure, insert a line near the beginning which executes the procedure from (2), above, loading all the procedures needed:

```
gc_libr
```

log (destination)

If you want to have a record of everything which appears on the screen (as a debugging tool, for example), use the "log" command. It creates a file on disk which contains all screen output. For example, to create a file called monday.log, type:

```
log monday.log
```

and press Enter. When you want to turn off the log function, type:

```
log con:
```

and press Enter.

Device Commands

use (device name)

When the system contains more than one device of a given type, procedures must specify which one is to be used. The "use" statement names the device which the system is to use by default until it encounters another "use" statement. If there were two robots which had been named robot_manny and robot_moe in the System Configuration Utility, a procedure might contain the line

```
use robot_moe
```

near the beginning. All subsequent commands for a robot would go to robot_moe. If a later subroutine were to use the other robot, the statement

```
use robot_manny
```

would signal the change to the system.

> **Note:** Syringe commands are associated by the system with the name of the particular syringe to which they were taught. Thus you do not need to use the "use" command for syringes. However, if you change the module name, the command may not function properly.

init (device name)

The "init" command causes the device named in the command to be initialized, following the initialization sequence established for that device in the system configuration file; for example,

```
init robot_ramon
```

display devices

This command generates a screen display of the configurations of all the devices listed in the system configuration file.

Robot Control

```
open

close
```

These two commands open and close the robot grippers.

> **Important:** There is also a "close (filename)" command in PERL. Be sure to always include the filename when you want to close a file, because "close" alone will close the grippers. On the other hand, if you try to use "close grippers" as a robot control command, the system will search for a file named "grippers".

```
up (distance)

down (distance)

relative (x,y,z,p,r)
```

The "up", "down", and "relative" commands all move the robot a specified amount relative to its present position.

> **Important:** Be sure that the destination you specify with "up", "down", or "relative" is within the work envelope of the robot. If you attempt to move the robot to a destination it cannot reach, it will reset; the grippers will open, releasing any container being grasped.

"Up" and "down" move the robot in the Z direction only, retaining the current X and Y coordinates and the pitch and roll. The distance is in millimeters, and can be either a constant or a variable. For example,

```
up 5
```

moves the robot up 5 millimeters from its present position.

```
down b
```

moves the robot down b millimeters. The distance argument can be either positive or negative; "down -4" has the same meaning as "up 4".

If you want the x and y coordinates or the pitch and roll to change, use the "relative" statement. The syntax rules for this command are as follows:

* The relative changes in x, y, and z are in millimeters; changes in wrist pitch and roll are in degrees.

* Either constants or variables are allowed.

* If you do not want to change all the coordinates, use zeros for the fields which will not change; for example,

  ```
  relative 0,5,0,0,10
  ```

* You must include all the coordinates to the left of the one(s) you want to change. You can drop the ones to the right. For example,

  ```
  relative 0,6
  ```

  would move the robot 6 millimeters in the +y direction.

* The commas separating the fields are required.

```
continuous
    position1
    position2
    position3
end continuous
```

The "continuous" command moves the robot through a series of positions without ramping speeds and stopping at each one. Only robot commands can appear in the "continuous" block, and they must all be positions; that is, you can not use a rack or grip pressure command in the block. You may, however, use any number of position commands.

Warning: Do not use "continuous" with commands which would reverse the robot arm motors at high speed. This could damage the robot. You may need to perform preventive maintenance more frequently if you use "continuous" statements.

```
speed (integer or variable)
```

You can control the relative speed of the robot by entering an integer or variable from 0 to 9. The slowest speed is 0, the fastest is 9.

Timer Statements

```
Set timer (no.) for (quantity of time) (unit of time)

wait for timer (no.)
```

The PERL software supports 20 internal timers for use in carrying out robotics procedures. You may set these timers individually, by number, for the desired length of time in either minutes or seconds. For example,

```
set timer 1 for 5 minutes

set timer 2 for 60 seconds
```

There is, in effect, no limit on the amount of time you can set, expressed in minutes or seconds. You can also use variables for the timer numbers and times, provided the values of the variables are within the range of valid numbers:

```
set timer n for x seconds
```

If the system is to do nothing else until the time has elapsed, the "set timer" statement would be followed by

```
wait for timer 1
```

or

```
wait for timer n
```

On other occasions, certain actions may be performed until the time elapses. In that case, you could use the "while" or the "until" statement (see the next section).

```
set timer 1 for 30 seconds
mix_samp
while timer(1) = 1
     cursor 10,10
     display "Still mixing"
end while
get_sample
```

When there is unexpired time remaining, the timer is equal to 1. When the time has elapsed, the timer equals 0. The "while"..."end while" statements in the above procedure are equivalent to "wait for timer 1".

The following timer statements both begin loops which depend on the timer being unequal to zero:

```
if timer(3)
while timer(3)
while timer(3) = 1
```

These statements mean if or while timer 3 has not expired (is unequal to zero).

Co-ordinated Procedures: Until and While

```
until (condition)

    end until

while (condition)

    end while
```

Sometimes different actions are dependent on each other; one action is to be carried out while a certain condition is true, or until another action is completed. In these situations, use the "until" or "while" statements. The following series of statements could be used to have the robot weigh and process samples for as long as there were any samples in the rack:

```
until (samples > 0) and (samples < 41)
     input "enter the number of samples to process:",samples
end until
until samples = 0
     get_samp
     weigh
     process
     samples = samples - 1
end until
```

The procedure segment which follows uses the "while" statement. A major difference between "until" and "while" is that the system terminates execution of the body of the "while" as soon as the condition is no longer met; the body of an "until" will be carried out at least once before it is terminated.

```
until (samples > 0) and (samples < 41)
     input "enter the number of samples to process:", samples
end until
while samples > 0
     get_samp
     weigh
     process
     samples = samples - 1
end while
```

You can use numeric and string functions from Tables 4-4 and 4-5 as conditions in "until" and "while" statements; for example,

```
while sqrt(x) < 100
```

All "while" and "until" blocks must be terminated by an appropriate "end" statement, as seen in the examples above.

Input/Output to the Monitor and Keyboard

```
display (desired output)
cursor (h line no.),(v column no.)
clear
```

As a PERL procedure is running, it may use the monitor to give the operator information such as the number of the sample being prepared. It may also prompt the operator when human action is needed. Such monitor displays are initiated by "display" statements within the procedure. (This is similar to the PRINT statement in BASIC.)

The argument of a "display" statement may be a constant, as in

```
display "End of UV sample preparation."
```

or a variable, such as

```
display sample_number
```

In the latter case, the monitor would display the actual number presently assigned to the variable sample_number.

The statement may contain a combination of constants, variables, and expressions, separated by semicolons. At execution, the semicolons prevent the cursor from advancing between each element of the display. The next constant, variable, or expression immediately follows the one which precedes it. This statement

```
display "Sample number "; sample_number; " of "; total_samples
```

would give the following result if the robot were at the twenty-eighth of forty samples:

```
Sample number 28 of 40
```

Note that quotation marks are necessary around the string constants.

You may tab between items on a displayed line, if you wish. The monitor screen is divided into vertical zones by tabs set every 14 spaces (at columns 14, 28, 42, etc.). If you separate the variables and constants in a "display" statement by commas, rather than semicolons, each item will begin at the beginning of the next zone. This feature is useful for printing columns of data. The lines

```
display "sample no.", "mass", "vol"
display sample_num, samp_wt, samp_vol
```

could be used as part of a procedure to generate a table of data:

```
sample no.      mass            vol

1               0.025           2.51
2               0.023           2.48
```

Displays will begin by default at the present cursor location. However, you can place the display wherever you wish by means of the "cursor" statement. The monitor screen is 80 vertical columns wide and 23 horizontal rows high. The upper left corner of the screen is cursor position 1,1; the lower right is position 23,80. The statement

```
cursor 5,1
```

will place the cursor on the first space of the fifth line. You may use an expression in place of a line number; the system will evaluate the expression and place the cursor accordingly:

```
cursor ((2*x)-3),2*y
```

As a procedure proceeds, messages will accumulate on the screen. The "clear" statement, which requires no arguments, allows you to remove all the accumulated displays from the screen.

input [prompt] (variable)

The "input" statement allows the operator to type in information from the keyboard as a procedure is running. When the system encounters an "input" statement, it waits for an entry. The value of the typed entry (numeric or character) is assigned to the variable in the "input" statement.

```
input num_samples
```

would result in the question mark prompt. The typed entry **25**, followed by a carriage return to signal the end of the response, assigns the value 25.0000 to the variable num_samples.

It is usually helpful to display a prompt which tells the operator what sort of response the system expects. The previous example could have used the statement

```
input "Total number of samples"; num_samples
```

Only one prompt and one variable response are allowed in each "input" statement.

Note that the statement in the preceding paragraph is similar in format to a "display" statement which has a character constant in quotation marks, followed by a semicolon, then a variable name. "Input" statements, like "display" statements, can also use a comma instead of a semicolon. As shown, the statement will cause the system to query

```
Total number of samples?
```

and wait for a response. If you use a comma instead of a semicolon, you suppress the question mark.

```
input "Total number of samples:", num_samples
```

will produce

```
Total number of samples:
```

```
display "\(octal)"

display (variable)
```

The entire IBM PC character set is available to you from PERL. You can display Greek letters, double lines, arrows, and many other non-keyboard characters.

Each character in the set is identified by an octal number, as shown in Table 4-6.

To access a character, use the \ symbol followed by the octal identifier, all surrounded by quotation marks. For example,

```
display "\101"
```

will display the character "A".

TABLE 4-6
THE IBM PC CHARACTERS AND THEIR OCTAL CODES

41	!	116	N	173	{	250	¿	325	╒
42	"	117	O	174	¦	251	⌐	326	╓
43	#	120	P	175	}	252	¬	327	╫
44	$	121	Q	176	~	253	½	330	╪
45	%	122	R	177		254	¼	331	┘
46	&	123	S	200		255	¡	332	┌
47	'	124	T	201		256	«	333	█
50	(	125	U	202		257	»	334	▄
51	)	126	V	203		260	░	335	▌
52	*	127	W	204		261	▒	336	▐
53	+	130	X	205		262	▓	337	▀
54	,	131	Y	206		263	│	340	α
55	-	132	Z	207		264	┤	341	ß
56	.	133	[	210		265	╡	342	Γ
57	/	134	\	211		266	╢	343	π
60	0	135	]	212		267	╖	344	Σ
61	1	136	^	213		270	╕	345	σ
62	2	137	_	214		271	╣	346	µ
63	3	140	`	215		272	║	347	τ
64	4	141	a	216		273	╗	350	Φ
65	5	142	b	217		274	╝	351	Θ
66	6	143	c	220		275	╜	352	Ω
67	7	144	d	221		276	╛	353	δ
70	8	145	e	222		277	┐	354	∞
71	9	146	f	223		300	└	355	Ø
72	:	147	g	224		301	┴	356	ε
73	;	150	h	225		302	┬	357	∩
74	<	151	i	226		303	├	360	≡
75	=	152	j	227		304	─	361	±
76	>	153	k	230		305	┼	362	≥
77	?	154	l	231		306	╞	363	≤
100	@	155	m	232		307	╟	364	⌠
101	A	156	n	233		310	╚	365	⌡
102	B	157	o	234		311	╔	366	÷
103	C	160	p	235		312	╩	367	≈
104	D	161	q	236		313	╦	370	°
105	E	162	r	237		314	╠	371	∙
106	F	163	s	240		315	═	372	·
107	G	164	t	241		316	╬	373	√
110	H	165	u	242	ó	317	╧	374	ⁿ
111	I	166	v	243	ú	320	╨	375	²
112	J	167	w	244	ñ	321	╤	376	■
113	K	170	x	245	Ñ	322	╥	377	
114	L	171	y	246	ª	323	╙	400	

You can embed an octal code in a string constant:

```
display "Proposed \341 Test Sites"
```

will display

```
Proposed ß Test Sites
```

You can also assign the octal code to a string variable; for example,

```
char_b$ = "\341"
display char_b$
```

The two above statements will display the character "ß".

redirect input [from] (device)

redirect output [to] (device)

The standard input device for the MasterLab System is the keyboard of the System Controller. The standard output device is the screen. If you wish to have the System Controller accept input from a device connected at port com1:, and you had named the port "lims" in the System Configuration File, you could issue the command

```
redirect input from lims
```

Input can also come from a file.

```
redirect input from "datafile"
```

Similarly, output could be redirected to the printer, a port name, or to a file.

```
redirect output to "newfile"
```

When you want the input or output to again come from or to the standard device, you must use another "redirect" command. In the following examples, "con:" refers to the console (screen and keyboard):

```
redirect input from con:
redirect output to con:
```

inkey$

When the system encounters a statement containing "inkey$", there is a pause in procedure execution; the pause lasts until the operator signals by hitting a key. The following two lines accomplish this:

```
display "Press any key to continue."
x$ = inkey$
```

The string value of the key hit will be assigned to x$. (Note that inkey$ itself is never assigned a value.)

The pause may provide an opportunity for the operator to make a decision:

```
display "Do you wish to change solutions? Type y or n."
x$ = inkey$
if x$ = "y" then
    input "Enter concentration of new solution"; conc_soln
end if
prep_sam
```

The above procedure segment provides a pause in which the operator can change solutions, if desired. If the value of x$ indicates that this has been done, input of information is requested before the procedure continues with the sub-procedure "prepare_samples". (See the description of the "if" statement.)

readkey$

The "readkey" keyboard function provides a related facility for operator control. It allows the procedure to continue without pause unless a key has been pressed since the last time the keyboard buffer was read. If no key has been pressed, the value of the readkey function is the null string, "".

The following procedure will continue to process samples until the operator presses a key. The system will not save the value of the key pressed.

```
procedure rdky_dem
    while readkey$ = ""
        display "Press any key to stop"
        process
    end while
end procedure
```

If the value of the key pressed is important, use readkey$ as in the following:

```
procedure rdky_sav
    until x$ <> ""
        display "Press a key to stop"
        process
        x$ = readkey$
    end until
    display "The key pressed was "; x$
end procedure
```

Printing

```
print (desired output)

    print screen
```

The "print" statement functions exactly like the "display" statement, except that its output goes to the printer rather than the monitor screen. See the discussion of "display" for a description of the varieties of syntax available.

The "print screen" statement causes the entire contents of the screen to be output to the printer. When you are in the Direct Command Processor, you can accomplish the same thing by pressing the shift key and the PRint SCReen key simultaneously. (The printer must be turned on.)

The pause may provide an opportunity for the operator to make a decision:

```
display "Do you wish to change solutions? Type y or n."
x$ = inkey$
if x$ = "y" then
    input "Enter concentration of new solution"; conc_soln
end if
prep_sam
```

The above procedure segment provides a pause in which the operator can change solutions, if desired. If the value of x$ indicates that this has been done, input of information is requested before the procedure continues with the sub-procedure "prepare_samples". (See the description of the "if" statement.)

readkey$

The "readkey" keyboard function provides a related facility for operator control. It allows the procedure to continue without pause unless a key has been pressed since the last time the keyboard buffer was read. If no key has been pressed, the value of the readkey function is the null string, "".

The following procedure will continue to process samples until the operator presses a key. The system will not save the value of the key pressed.

```
procedure rdky_dem
    while readkey$ = ""
        display "Press any key to stop"
        process
    end while
end procedure
```

If the value of the key pressed is important, use readkey$ as in the following:

```
procedure rdky_sav
    until x$ <> ""
        display "Press a key to stop"
        process
        x$ = readkey$
    end until
    display "The key pressed was "; x$
end procedure
```

Printing

```
print (desired output)

print screen
```

The "print" statement functions exactly like the "display" statement, except that its output goes to the printer rather than the monitor screen. See the discussion of "display" for a description of the varieties of syntax available.

The "print screen" statement causes the entire contents of the screen to be output to the printer. When you are in the Direct Command Processor, you can accomplish the same thing by pressing the shift key and the PRint SCReen key simultaneously. (The printer must be turned on.)

Making Decisions within Programs

```
if (expression) then
        statement(s)
[else
        statement(s)]
end if
```

When the system encounters a statement beginning with "if" and an expression, it evaluates the expression to see if it is true or false. If the expression is true, the statements following "then" will be executed. If it is not true, the statements following "else", if any, will be executed. The following examples of "if" statements have a variety of expressions to be evaluated:

```
if timer (1) = 0 then
if instrument_ready = 0 then
if instr(a$,b$) = 0 then
if sample_wt <= 0.5 then
```

The first example refers to whether the time set on timer 1 has elapsed. The second refers to a condition which would have been previously defined in terms of a switch being on or off. The third tests the condition of a string function from Table 4-5. The last involves relational operators. The operators allowed in PERL are:

=	equal to
<>	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
AND	logical AND
OR	logical OR
NOT	logical NOT

For example,

```
if b < a or b > c then
if a = b AND a = c then
```

Note that logical operators can be used only with numeric variables, not with strings. Compare the following legal and illegal uses of logical operators:

Legal	Illegal
if n = a and t = b then	if n$ = "a" and t$ = "b" then
if n = a and t = 1 then	if n$ = "a" and t = 1 then

Use the "else" portion of the if block only if action is to be taken when the expression is false that would not be taken if it were true. For example,

```
if sample_wt <= 1.0 then
     add_solv
     mix_samp
else
     dscd_sam
end if
```

```
case (var) or case (expr)

[default]

end case
```

The "case" statement provides another way of specifying alternative courses of action. Use it when several different alternatives are possible, depending on the value of a variable or expression. As an example, suppose that three different procedures could be called, depending on the volume of the sample. The following "case" construction specifies the alternatives:

```
case vol_sample
     is 10:
          prp_10ml
     is 5:
          prep_5ml
     is 1:
          prep_1ml
end case
```

If the value of vol_sample were 10, the procedure prep_10ml would be executed. Otherwise, the appropriate other procedure would be used.

The lines which follow the "case" statement must consist of:

the keyword "is" with either a variable or a constant (not an expression) before a colon;

a statement (or statements) on the line(s) following the "is".

In the following example, there are two statement lines after each "is" line:

```
total_vol = weigh_empty_tube
get_samp
vol_sample = weigh_sample
case vol_sample
     is 10:
          prp_10ml
          total_vol = total_vol + 10
     is 5:
          prep_5ml
          total_vol = total_vol +5
end case
```

If the "case" block contains two or more "is" lines in succession, PERL interprets them as if they were connected by a logical "or". For example,

```
case name$
     is "Fred":
     is "Sam":
          proc_men
     is "Harriet":
          proc_wmn
end case
```

In the above procedure, the system would execute "proc_men" if name$ were either Fred or Sam.

The optional "default" statement is equivalent to an "else." If all the other conditions fail, the system will execute the default. For example,

```
case i
     is 1:
          display "I is 1"
     is 2:
     is 3:
          display "I is 2 or 3"
default:
          display "I is not 1, 2, or 3"
end case
```

In the example, the default display "I is not 1, 2, or 3" would appear if i were anything other than 1, 2, or 3.

Repeated Actions: Loops

```
for (var) = (exp1) to (exp2) [step (expr)]
     statement(s)
next (var)
```

When the same steps are to be carried out several times in succession in a procedure, it is wasteful of programmer time and computer memory to repeat the program lines involved. Instead, the repeated lines are enclosed in a "program loop". The loop begins with a "for" statement which indicates how many times the action is to be repeated. It ends with a "next" statement. In the following example, the variable i is used simply as a counter; its value is increased by 1 each time the loop is executed, until i equals 5. Then the statement following the loop is executed.

```
for i = 1 to 5
     display "Happy"
next i
display "Birthday"
```

The output generated by this program would be

```
Happy
Happy
Happy
Happy
Happy
Birthday
```

The value of the counter variable can also be used within the body of the loop. The following procedure would display the numbers from 1 to 100:

```
procedure a_loop
for a = 1 to 100
     display a
next a
end procedure
```

If you want to increment the counter variable by more than one, or to use negative increments, use the optional "step" argument:

```
procedure countdn
display "Countdown:"
for i% = 10 to 0 step -1
    display i%
next i%
display "BLAST OFF"
end procedure
```

The above procedure would produce the following display:

```
Countdown:
10
9
8
7
6
5
4
3
2
1
BLAST OFF
```

You may nest one "for...next" loop inside another. The following program will display the multiplication tables up to the twelfth. The "display" statement causes the program to skip a line between successive tables:

```
procedure mult
for i = 1 to 12
    for j = 1 to 12
        display i; "*"; j; " = "; i*j
    next j
    display
next i
end procedure
```

Note that the "for j" statement comes <u>after</u> the "for i", and the "next j" comes <u>before</u> the "next i", so that the <u>j loop</u> is completely enclosed within the i loop. Up to 16 loops may be nested in this manner.

"For" loops allow easy access to the individual elements of an array. If the array subscript variable is the same as the counter variable of the loop, the value of the subscript changes each time the loop is executed.

```
procedure get_mass
dim weight(10)
for i = 1 to 10
    read weight(i) from "weights.dat"
    display i; " = "; weight(i); "g"
next i
close "weights.dat"
end procedure
```

As it reads weights.dat in the above example, the computer generates a display similar to the following:

```
1 = 0.543g
2 = 0.601g
3 = 0.522g
4 = 0.576g
5 =            etc.
```

Be aware that every time you use arrays, you must use the "dim" statement described below. The system does not assign any default subscripts.

dim (var(i))

When you use array variables, you must allocate computer memory for storing them. This is accomplished by the "dim" statement, which is usually placed near the beginning of the procedure, and must always come before the first use of the array variable. In the procedure above which read 10 weights from weights.dat, for example, the statement

```
dim weight(10)
```

would have to occur before the first use of the array variable. "dim" stands for dimension. The statement not only allocates memory for the weight(i) array, it tells the system that the largest subscript will be 10, and it initializes all the array variables to 0. (String variables are initialized to "", the null string.)

You may dimension more than 1 array in a single statement. The statement

```
dim weight(10), month%(12), name$(6)
```

dimensions two numeric arrays and one character array, allocating space to store ten weights, twelve months, and six names.

Arrays may have multiple dimensions, up to three. Such arrays are called two-dimensional or three-dimensional. A two-dimensional array dimensioned

```
dim array(2,3)
```

would contain the variables

```
array(1,1)    array(1,2)    array(1,3)
array(2,1)    array(2,2)    array(2,3)
```

The following nested "for...next" loops use such an array:

```
oldfile$(1) = "oldfile1"
oldfile$(2) = "oldfile2"
for i = 1 to 2
    for j = 1 to 3
        read info$(i,j) from oldfile$(i)
    next j
next i
```

The above procedure would read the first three pieces of information from each of two files, called oldfile1 and oldfile2.

Warning: The software will allow you to dimension an array for up to three dimensions of 256 elements each (subscripts 0 to 255). Be aware, however, that you may not be able to store large arrays in the computer, because of memory limitations. Arrange your program so that you store the array on disk, and retrieve elements as you need them.

break

The "break" statement transfers control out of a loop and gives control to the next statement following the loop. For example,

```
num_of_samples = 40
while num_of_samples > 0
    get_samp
    way_samp
    weight = get_weight
    if weight = 0 then
        break
    end if
    process
    num_of_samples = num_of_samples - 1
end while
data_prs
```

In the above example, if the sample weight is 0, then the procedure does not continue with the "process_sample" statement. Instead, it exits from the "while" loop and continues with "data_prs".

If you use a "break" statement within nested loops, the break applies only to the innermost of the loops presently executing. Control will continue at the first statement outside that loop. In the example which follows, two loops are executing. "Break" removes control from the "for...next" (innermost) loop, but not the "while" loop.

```
num_of_samples = 40
while num_of_samples > 0
     get_samp
     way_samp
     weight = get_weight
     for i = 1 to 5
          move_to_syr
          fill_1ml
          way_samp
          weight = get_weight
          if weight = 1.1 then
               break
          end if
          pr_nusam
     next i
     num_of_samples = num_of_samples - 1
end while
data_prs
```

In the preceding example, successive 1 ml aliquots are removed from the sample tube within the "for...next" loop. The "if" statement tests to be sure there is 1 ml remaining in the tube for the next aliquot. If not, control is transferred out of the "for...next" loop, to the statement following the "next i". However, control is still within the "while" loop.

Input/Output Via RS232 Ports

```
define (var) as (port name)

send (var)

receive (var)

input$ (device name, no. chars)
```

PERL permits serial communication with external devices connected to the RS232 ports.

Defining a buffer

Whether the communication is to be input or output, the port must first be opened by means of the "define" statement. The port name argument of the statement must be a port which has been defined in the System Configuration File. The "define" statement sets aside a buffer whose name is a string variable. This buffer will contain the input or output data; it may contain up to 255 characters.

In the example which follows, LAMBDA3 is the name assigned to the port in the System Configuration File; the buffer is to be named a$:

```
define a$ as LAMBDA3
```

Outputting Data

After the procedure has defined the buffer, the desired output is assigned to it ; for example,

```
a$ = "GET ABSORBANCE"
```

Then the output can be sent:

```
send a$
```

Inputting Data

The statement "receive a$" will allow the buffer to receive communication from the device at LAMBDA3. Thus the entire sequence of statements for obtaining an absorbance from the Lambda-3 instrument would be as shown on the following page.

```
procedure get_abs
define a$ as LAMBDA3
a$ = "GET ABSORBANCE"
send a$
receive a$
display a$
end procedure
```

Receiving Input without Terminators

Usually a serial input ends with a terminator that signals the end of the transmission. The input$ function allows a specified number of characters to be transmitted without a terminator.

The syntax for using the input$ function is as follows:

```
a$ = input$ (device name, number of characters)
```

The device name is the name assigned to the device in the system configuration file. The comma and the parentheses are required.

As an example, suppose you wanted to input three characters from a Lambda 4C spectrophotometer which was named "L4C" in CONFIG. The string which is input is to be assigned to the variable d$. The procedure would contain the line

```
d$ = input$ (L4C, 3)
```

The number of characters must be more than zero, and less than or equal to 512.

Using Files on Disk

```
 write (var) to (filename)
append (var) to (filename)
read (var) from (filename)
     rewind (filename)
      close (filename)
```

PERL uses sequential data files to store data, such as sample weights acquired during automated sample preparation procedures, on disk.

You may write data to a new file or append it to an existing one. The stored data can later be read from the file and processed by PERL procedures or by other software available for the IBM PC.

The rules for using file names in PERL are as follows:

* The filename must have no more than 8 characters.
* When used as an argument in a command, the filename must be in quotes.
* The filename argument in a command may be a string constant, such as datafile$.
* PERL supports use of DOS path names for multiple directories on a single disk. For example, to write data to a file called "july" on a directory called "data" which is immediately off the root, use

  ```
  write a$ to "\data\july"
  ```

  See Chapter 5 in your DOS manual for an explanation of directories and path names.
* You can use extensions (for example ".dat") on filenames if the file is not to be compiled.

There is no "open (filename)" statement in PERL; the software will open the file automatically when necessary. You must close the file with a "close (filename)" statement, however.

> **Important:** The "open" and "close" commands (without a filename) open and close the robot grippers. Do not use these words by themselves unless that is what you intend.

The following procedure segment would store sample weights in a new file called mysampls:

```
for i = 1 to num_samples
    way_samp
    write sampwt(i) to "mysampls"
    prep_sam
next i
close "mysampls"
```

(It is assumed that way_samp and prep_sam are existing procedures, and sampwt() is dimensioned.) Because this is a sequential file, the sample weights are written into the file in order, separated by carriage returns; that is, the first record in the file would correspond to the first sample weighed, etc. If you later wanted to file the weights of another batch of samples into the existing file mysamples, the statement

```
append sampwt(i) to "mysamples"
```

would be needed. This would add the additional weights to the end of the file.

After the system reads from a file, it must "rewind" the file (figuratively) to its beginning before it can read the contents again. The following procedure reads yourfile, rewinds the file, then reads the data again for a different purpose.

```
dim data(50)
for i = 1 to 10
    read data(i) from "yourfile"
next i
rewind "yourfile"
press_da
for i = 1 to 50
    read data(i) from "yourfile"
next i
close "yourfile"
```

Note that, because PERL files are sequential, you can not read just the one record in which you may be interested. To read the twenty-fifth record, for example, you must read records 1 through 25.

eof(var)

eof("filename")

On many occasions, you as programmer will not know how many records a file contains. You therefore will not be able to specify how many records are to be read. The End of File function, eof, is provided to deal with that situation. In the following procedure, a very large number of records is specified. After each is read, the procedure checks to see if it is the last in the file. When the last record is reached, a "break" statement transfers control out of the loop.

```
dim data(200)
for i = 1 to 200
     read data(i) from "datafile"
     if eof("datafile") = 1 then
          num_data = i
          display "File contains "; i; " records"
          break
     end if
next i
for i = 1 to num_data
     press_da
next i
```

The function eof is equal to 0 when there are records remaining to be processed; eof equals 1 at the end. Thus you could also process records in a "while" or "until" block:

```
until eof(a$) = 1
```

or

```
while eof("myfile") = 0
```

Date and Time Statements

date$

day$

month$

time$

The "date$", "day$", "month$", and "time$" statements cause the present date, day, month, or time to be output to the screen. The following procedures, run at noon on December 20, 1985, demonstrate the formats in which the time statements output data:

```
a$ = date$
display a$
    12/20/1985

b$ = day$
display b$
    Friday

c$ = time$
display c$
    12:00:00

d$ = month$
display d$
    December
```

These statements are useful for dating files created with PERL procedures. For example, after creating a file called "newdata", use the statements

```
a$ = date$
b$ = time$
append a$ to "newdata"
append b$ to "newdata"
```

to add the date and time of creation to the file on disk.

5 THE PERL EDITOR

5 THE PERL EDITOR

INTRODUCTION

You will write your PERL applications programs (called procedures) using the PERL Editor, a full screen editor which you can access from PERL by means of the F2 function key of the System Controller.

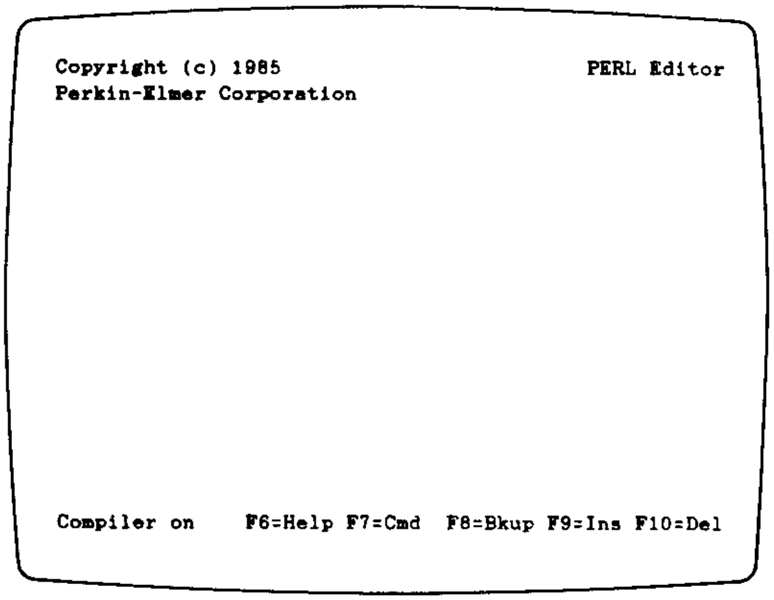
You can use the Editor to create and edit either procedures or text files. Text files are sequential files created for use in a PERL procedure. Such a file may, in fact, have actually been created during execution of a PERL procedure; you would then use the Editor for editing only.

The Editor functions differently if you are editing a procedure, rather than a text file. The system compiles a procedure into an intermediate form, ready to be interpreted when it is run. This permits more rapid execution.

The PERL Editor uses the System Controller keyboard keys for cursor movement and such editing functions as inserting characters. A menu, accessed by a function key, is available for choosing among major tasks such as creating a procedure or saving a procedure on disk.

EDITOR COMMANDS

The PERL Editor is accessed by the F2 function key. Once you have entered the Editor (see "Entering the Editor" for detailed information on doing this) the screen will be similar to the following, with a cursor positioned at the upper left corner, and prompts visible for the five function keys F6 to F10.



```
Copyright (c) 1985                                PERL Editor
Perkin-Elmer Corporation




Compiler on     F6=Help F7=Cmd  F8=Bkup F9=Ins F10=Del
```

Figure 5-1 - PERL Editor, Initial Screen

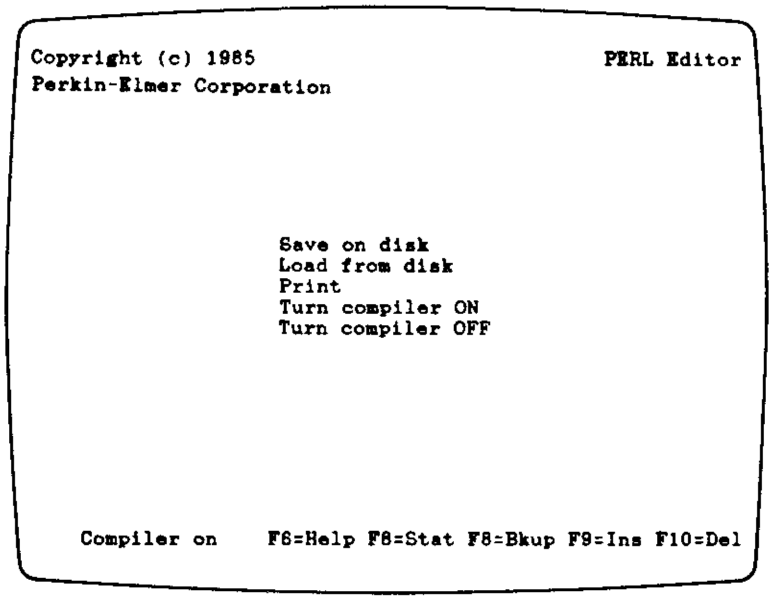
You may select Editor commands either from the menu obtained by pressing F7, or by using certain typewriter keys. The remainder of this section describes both types of commands.

Commands on the Editor Menu

When you press F7 (Edit Commands) while in the Editor, the menu shown in Figure 5-2 appears at the right of the screen. To select a command from the menu, use the up and down arrow keys to highlight the desired command, then press the Enter key. After the command has been executed, the menu will be erased, and the text will return to the screen.

The following commands are available:

Save on Disk	Stores the file in memory on disk under the name provided. Procedures will be stored in both ASCII and compiled form, text files in ASCII.
Load from Disk	A prompt will request the name of the file to be loaded. The new file will replace any file already in memory. A backup copy of the file being loaded will be placed on disk, with the name "FILENAME.BAK".
Print	Sends the entire file in memory to the printer.



```
Copyright (c) 1985                                  PERL Editor
Perkin-Elmer Corporation

                     Save on disk
                     Load from disk
                     Print
                     Turn compiler ON
                     Turn compiler OFF

   Compiler on    F6=Help F8=Stat F8=Bkup F9=Ins F10=Del
```

Figure 5-2 - Edit Commands Menu Screen

Turn Compiler On	Causes the system to automatically compile procedures to an executable form when you save them to disk.
Turn Compiler Off	Turns off automatic syntax checking.

Commands Controlled by Keyboard Keys

Key	Command
F6	Help

Note: The HELP function has not yet been implemented.

F7	Edit Commands: Brings up the Commands Menu described previously. When the Commands Menu is up, this function key becomes Quit: Leave Edit Commands Menu.
F8	Bkup: Method for leaving the Editor.
F9	Insert Line: Creates a blank line in the text above the present cursor location, in which additional text can be written.
F10	Delete Line: Deletes the line of text in which the cursor is residing.

Left Arrow	Moves cursor to the left.
Right Arrow	Moves cursor to the right.
Up Arrow	Moves cursor up.
Down Arrow	Moves cursor down.
Home	Moves cursor to the beginning of the line of text.
End	Moves cursor one space to the right of the last character on the line.
PgUp	Brings up the previous page of text.
PgDn	Brings up the next page of text.
Del	Deletes the character at the present cursor position. All characters to the right of the cursor move one place to the left.
Ins	Allows characters to be inserted ahead of the cursor. Press the key again to return to normal operation.
Tab	Moves cursor to the next tab stop. Tabs are set every 8 spaces. Spaces, rather than tab characters, will be used in the text.
Back Arrow	Backspace key. Moves cursor to the previous character.

PROGRAMMING A PROCEDURE

Accessing the Editor

There are four possible reasons for using the Editor; it is used either to create a new file or to edit an existing one, and the file may be either a procedure or a text file. Your intended purpose affects the way in which you enter the Editor.

- To create a new file, either a procedure or a text file:
  Press F2.
  Check the display at the bottom of the screen to see whether the compiler is ON or OFF. Press F7 and use the commands menu to change the ON/OFF status, if necessary.
  Press Enter.

- To edit an existing text file:
  Press F2.
  Press F7.
  Highlight Load from Disk; when prompted, give the filename.
  Press Enter.
  If the compiler is on, use the commands menu to turn it off.

- To edit an existing procedure file:

  Press F2.
  Press F7.
  Highlight Load from Disk; when prompted, give the filename.
  Press Enter.
  If the compiler is off, turn it on.

Editing a Procedure

The section which follows directs you to access the PERL Editor, type in a sample procedure, and then modify it using the Editor commands. Doing so will allow you to experience the editing process for yourself.

1) Using the procedure given above for entering the Editor to create a new procedure, access the PERL Editor.

2) Type the following procedure, one line at a time, pressing Enter after each line. You can type in upper case, lower case, or mixed cases.

   If you make an error before pressing Enter, you can backspace to correct it. If you press Enter before correcting the error, use the procedure in (3), on the next page, to correct it.

   <u>**Note:**</u> Line numbers are included for convenience in the following discussion. They are not used in PERL procedures.

```
 1   procedure mix_samp
 2        above_mixer
 3        turn_on_mixer
 4        down_5cm
 5        open_gripper
 6        up_3cm
 7        set timer 1 for 30 seconds
 8        turn_off_mixer
 9        down_3cm
10        close_gripper
11        above_mixer
12   end procedure
```

3) To change the 3cm distance in lines 6 and 9 to 2cm:

   a) Use the arrow keys to move the cursor to the 3 in line 6.

   b) Press the Del key. The 3 will be deleted.

   c) Press the Ins key and type **2.** A 2 will be inserted. Press Ins again to turn off the insert function.

   d) Repeat steps a-c for line 9.

4) Use the same procedure as in step 3 to change the 30 in line 7 to 15.

5) To insert a "wait for timer" statement after line 7:

   a) Place the cursor in line 8.

   b) Press the F9 (Insert Line) function key. A blank line will appear above line 8.

   c) Move the cursor to the blank line and type

      wait for timer 1 Enter

6) To change the "above_mixer" commands in lines 2 and 11 to "over_mixer":

   a) Press the F7 (Edit Commands) function key to bring up the Edit Commands menu.

   b) Highlight "Change String", and press Enter.

   c) In response to the prompts which will appear, type

      String to Change: **above_mixer**
      Replace with: **over_mixer**

   <u>**Note:**</u> It is a good idea to give the longest possible string to be changed. If, in the example above, you had typed only "above" to be changed, commands such as "above_rack" or "above_tube" would also have been changed throughout your procedure.

7) To move lines 4-6 so that they follow line 2:

   a) Place the cursor at the beginning of line 4. Press F7 to bring up the Edit Commands. Highlight "Mark Block", and press Enter.

   b) Repeat step a, placing the cursor at the end of line 6. Lines 4-6 will be marked.

c) Move the cursor to the beginning of line 3. Press F7 and highlight "Paste Block". Press Enter. The marked lines will be inserted ahead of the cursor.

d) Press F7 and highlight "Cut Block". The marked lines will be deleted from their original location.

The procedure should now appear similar to the following:

```
procedure mix_samp
    over_mixer
    down_5cm
    open_gripper
    up_2cm
    turn_on_mixer
    set timer 1 for 15 seconds
    wait for timer 1
    turn_off_mixer
    down_2cm
    close_gripper
    over_mixer
end procedure
```

You could save the procedure by highlighting "Save on Disk" on the Edit Commands menu. (If you save it, be sure to delete it afterward. To use the DOS command ERASE, see Appendix 2.)

Note: Procedure names, which you assign in the first line of the procedure, can contain no more than 8 characters. If more than 8 characters are present, you will receive the message

```
Procedure name exceeds 8 characters
```

when you attempt to save the procedure.

Editing a Text File

The editing process is exactly the same for a text file as for a procedure, except that there is no automatic compilation.

When you create a new text file, each record in the file should be typed on a separate line. You might, for example, type a list of identifying alphanumeric information to be printed with each sample number as the samples are processed.

A text file generated by a running procedure could contain sample weights and other quantitative information. You could call the file into the editor to make annotations, then print it using the "print" command on the Edit Commands menu.

Text files are saved by selecting "Save on Disk" and entering a file name when prompted. The compiler should be off.

Leaving the Editor

Use the F8 (Bkup) function key to leave the Editor and return to the PERL Main Menu Screen.

When you press F8, the screen will ask if you want to save your file. If you have already saved it, press Enter. If you have not, press **y**; the file will be saved (and compiled, if the compiler is on).

6 CREATING AN APPLICATIONS PROGRAM

6 CREATING AN APPLICATIONS PROGRAM

The first two parts of this section will take you through the process of planning and writing a PERL applications program (called a procedure). The planning is as important as the writing. Many of the statements in the procedures you write will be commands you create with the Teach Programs. This teaching is the most time-consuming part of robotics programming, and you will greatly improve your efficiency if you first invest time in carefully analyzing your procedures to determine the commands you need to teach.

Two places in this manual contain sample PERL procedures which you may be able to adapt to your own application:

* At the end of Section 6 are procedures for carrying out the tasks of each of the major modules of the MasterLab System.

* In Appendix 4 is an extensive procedure which integrates many features of PERL and makes use of several subprocedures.

The examples given in this manual are concerned with applications which specifically involve robotics. If you wish to create PERL programs which manage your data files, process your data, or are concerned with input/output, the PERL statements in Section 4 are available to you. These statements are similar to BASIC, and a BASIC textbook will provide help should you need it.

The data files which you may accumulate on disk while using the MasterLab System can also be processed by third-party software of your choice. You can not access such software from PERL, however. To leave PERL and return to DOS, type **system** while the PERL prompt is displayed. From DOS, load and use third-party software as directed in the manual for that software.

PLANNING THE PROCEDURE

Defining the Task

In general, the planning stage of PERL programming consists of analyzing the task the MasterLab System is to perform, and determining the individual steps in the task. You need to consider:

- What is the overall task to be performed?
- What are the smaller tasks involved in the overall task? These will each be directed by a procedure within the final procedure.

Important: A PERL procedure may be run from within another PERL procedure simply by using its name as a statement line. Thus a given procedure may function either as an independent program or as a subroutine. The term "procedure" will be used to cover both instances.

- What are the individual steps in the tasks, and which module carries out each step? These are the commands which you will need to create with the Teach Programs and use in your procedures.

As an example, the rest of this "Planning the Procedure" section will take you through the steps in planning the procedure for a gas chromatography sample preparation. The overall task is to dilute 0.2 ml of each sample with 1.8 ml of solvent, place 1 ml aliquots of the diluted samples in vials, seal the vials, and place them in an autosampler. Your first planning step is to identify the smaller tasks in this operation.

Identifying the Procedures Needed

In order to carry out the overall task defined in the last section, the system will have to perform the following tasks for each sample vial:

1. The robot gets a test tube of sample and brings it to the syringe.
2. The syringe draws in 1.8 ml solvent and 0.2 ml sample.
3. The robot returns the sample tube to the rack.
4. The robot picks up an empty test tube from another rack.
5. The robot takes the test tube to the syringe dispensing stand.
6. The syringe adds 0.2 ml of sample and 1.8 ml of solvent to the tube.
7. The robot takes the test tube to the mixer.
8. The mixer mixes the test tube contents.

9. The robot retrieves the tube from the mixer.
10. The robot takes the tube to the dispensing stand, where 1 ml of diluted sample is withdrawn.
11. The robot returns the test tube to its rack.
12. The robot picks up a sample vial and cap from the vial rack.
13. The robot takes the vial to the crimper, where the cap is left at the vacuum station.
14. The robot takes the vial to the dispensing stand, where the 1 ml aliquot is dispensed.
15. The robot takes the vial to the vacuum station and retrieves the cap.
16. The crimper seals the vial.
17. The robot places the vial in the autosampler.

It is wise at this point to compare your list of tasks with the list of existing procedures in your PERL Directory. (Use the F4 function key to display the Directory.) In the case of the example above, assume that your Directory already lists procedures for such standard tasks as:

task	task #	procedure name
mixing	7, 8, and 9	mix_samp
leaving vial cap at vacuum station	12 and 13	park_cap
crimping	15 and 16	crmp_vil
placing vials in autosampler	17	autosam
dispensing liquids (syringe plunger to 0.0 ml)	5 and 6; also 14	disp_liq

You would be able to use these procedures within your final procedure without change. Your planning can now focus on programming for tasks 1 through 4, 10, and 11.

Planning New Procedures

We assumed that there were six tasks in the above example which were not programmed in existing procedures. You could combine these into two procedures, one for tasks 1 through 4, one for tasks 10 and 11. Since both procedures are concerned with filling the syringe, after you write one of them, you can modify it to form the other.

tasks 1 through 4 - fill syringe: This procedure will load the syringe with the required amounts of sample and solvent for one vial. You could name it "fill_syr".

The steps involved in loading the syringe are:

1. The robot picks up a test tube of sample from the rack.
2. The robot brings the sample tube to the dispensing stand.
3. The syringe draws in 1.8 ml of solvent and 0.1 ml of air.
4. The test tube of sample is raised around the dispensing probe.
5. The syringe draws in 0.2 ml of sample.
6. The robot returns the test tube of sample to the rack.

In procedures involving the syringe, it is advisable to keep the syringe itself and its valves filled with solvent. Other fluids should be drawn into the dispensing tube, and then dispensed from that same tube, without having them pass through the valve or into the syringe. Different fluids should be separated in the tubing by drawing in a bubble of air between them. (See Figure 6-1.)

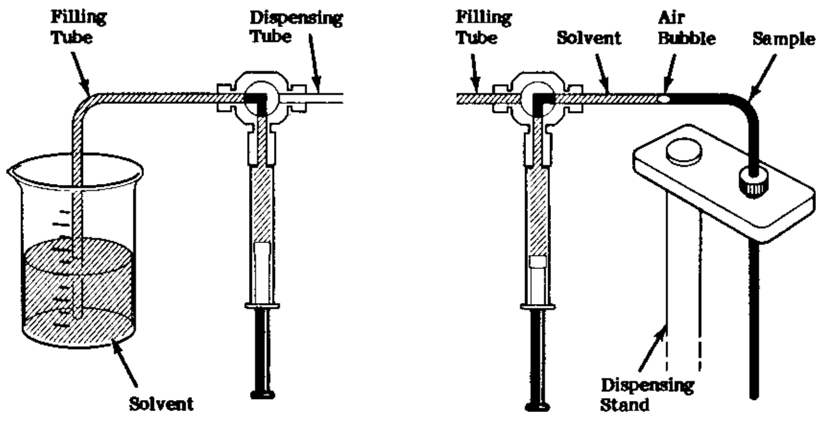


Figure 6-1 - Valve positions during Syringe Operation

In programming steps 1 and 6, assume that the robot Teach Program module called "Define a Rack" would have been used when the test tube rack was first installed in the system. (See Section 3 of this manual.) If the rack was named "sample", the robot can be directed to get a tube from the rack by using the command

```
sample i
```

where i is the number of the location of a particular tube in the rack. Similarly, returning the tube to the rack is accomplished by the command

```
sample i back
```

To program the rest of "fill_syr", you will need the following commands:

to take the test tube to the dispensing stand -	under_disp
to draw solvent and air into the syringe -	
with valve in input position, move plunger to 1.8 ml	fill_1.8_solvnt
with valve in output position, move plunger to 1.9 ml	fill_0.1_air
to raise the tube of sample -	at_dispenser
to draw sample into the syringe -	
with valve in output position, move plunger from 1.9 to 2.1 ml	fill_0.2_sample
to lower the sample test tube -	under_disp

Note: In actual practice, you should teach the robot several (at least 2) intermediate positions between "under_disp" and "at_disp". Otherwise, the robot hand will describe an arc between these two extreme positions, and the test tube may knock against and bend the dispensing probe.

tasks 10 and 11 - take an aliquot of sample: This procedure (called "aliquot") will be similar to "fill_syr", except that

it will not need to begin with picking up a tube
no solvent will be drawn in, although an air bubble will be
there will be 1.0 ml of sample drawn in, not 0.2
the rack to which the tube is returned will be called "empties"

To program aliquot you will need to create the following additional commands:

to draw air and sample into the syringe -	
with the valve in output position, move the plunger from 0.0 to 0.1 ml	make_bubble
with the valve in output position, move the plunger from 0.1 to 1.1 ml	fill_1_ml

The planning process is complete. You have a list of existing procedures which you can use, and a list of commands you must create before you can program the new procedures you need.

CREATING THE PROCEDURE

Teaching New Commands

Use the robot Teach Program and the syringe Teach Program to create the new commands you have listed which you will need in your procedure. Detailed instructions for using the Teach Programs are in Section 3 of this manual.

Writing Procedures

You will write each PERL procedure as an independent program by means of the PERL Editor. Be sure you familiarize yourself with the use of the Editor by carrying out the procedure in Section 5 of this manual.

Once you have used the Teach Programs to create all the commands you will need for your overall procedure, write your new procedures as follows:

1) From the PERL main menu, press the F2 (Edit) function key to enter the Editor. If the Edit Commands menu is not displayed, press F7.

2) If the compiler is not on, highlight "compiler on" in the menu, and press Enter.

3) Type the first line of the procedure, which should be a "procedure" statement:

   ```
   procedure fill_syr
   ```

   Note that the name given to the procedure does not exceed 8 characters.

4) Type the commands needed in your procedure, as you planned. Put one command on each line, pressing Enter after each.

5) The last line of the "fill_syr" procedure should be "end procedure". The procedure should now be similar to the following:

```
procedure fill_syr
sample i
fill_1.8_solvnt
fill_0.1_air
under_disp
at_dispenser
fill_0.2_sample
under_disp
sample i back
end procedure
```

7) Look over the program for places where statements from Section 4 are needed. For example, it would save time in executing the program if the syringe filled with solvent while the robot gets the test tube. "Parallel" could be inserted before line 2, and "end parallel" after line 3.

8) Insert appropriate "rem" statements to identify the function of the program and to explain any ambiguous statements. The final procedure will be similar to the following:

```
procedure fill_syr
rem Procedure fills dispenser with 1.8 ml solvent, 0.2 ml sample,
rem separated by 0.1 ml air.
rem
parallel
     sample i
     fill_1.8_solvnt
end parallel
fill_0.1_air
under_disp
at_dispenser
fill_0.2_sample
under_disp
sample i back
end procedure
```

9) Save your procedure by highlighting "Save on Disk" on the Edit Commands menu and pressing Enter.

10) Repeat steps 1 - 9 for all other procedures, such as "aliquot", which you have to create for your program.

Writing the Final Procedure

Writing the final procedure is similar to writing the shorter procedures it will use, except that most of the statements in the overall procedure will be names of other procedures to be executed.

It is advisable, where possible, to run the procedures individually before incorporating them into the overall procedure, to be sure that they work as you intended.

When you have written and tested all your procedures, write the main procedure as follows:

1) From the PERL main menu, press the F2 (Edit) function key to enter the Editor. If the Edit Commands menu is not displayed, press F7.

2) If the compiler is not on, highlight "compiler on" in the menu, and press Enter.

3) Type the first line of the procedure, which should be a "procedure" statement:

```
procedure gc_prep
```

4) Type the commands and procedure names needed in your procedure, as you planned. Put one command on each line, pressing Enter after each.

5) The last line of "gc_prep" should be "end procedure". The procedure at this stage should be similar to the following:

```
procedure gc_prep
fill_syr
disp_liq
mix_samp
aliquot
park_cap
disp_liq
crmp_vil
autosam
end procedure
```

6) Look over the procedure for places where BASIC-like statements from Section 4 are needed. The procedure above, as it stands, would only perform the procedure once. You must insert a "for...next" loop to repeat the procedure for the number of samples you have, 40 for example.

7) Insert appropriate "rem" statements to identify the function of the procedure and to explain any ambiguous statements. The final procedure will be similar to the following:

```
procedure gc_prep
rem Dilutes 0.2 ml sample with 1.8 ml solvent
rem Takes 1 ml aliquot to autosampler
rem
for i = 1 to 40
     fill_syr
     disp_liq
     mix_samp
     aliquot
     park_cap
     disp_liq
     crmp_vil
     autosam
next i
end procedure
```

8) Save your procedure by highlighting "Save on Disk" on the Edit Commands menu and pressing Enter.

9) Use the F8 (Bkup) key to leave the Editor.

10) Test your procedure by running it. To do this, from the PERL main screen type its name and press Enter, or press F1 (Run) and enter the name.

11) When your procedure is running correctly, link the needed subprocedures to the main procedure by using the "link" command; for example:

    link gc_prep

    This will create a file (in this case, "gc_prep.lnk") which has priority of execution over the corresponding compiled file ("gc_prep.cmp), and which will execute more rapidly because the linking is already done. However, if you want to change a procedure after it has been linked, you must either delete the lnk file from disk, or execute the "link" command again to overwrite the existing lnk file.

THE LIBRARY FACILITY

A major PERL procedure uses a number of other procedures as it runs. If the system loads each of these procedures from disk as it is needed, considerable run-time is devoted to waiting for procedures to load. You can avoid this problem by using the "library" facility, as follows.

1) Use the Editor to create a file which contains <u>all</u> the procedures you will need in your overall procedure. Do this by loading them into the Editor in sequence. The system will use the name of the first procedure you load as the library name.

```
procedure fill_syr
     .
     .
     .
end procedure
procedure disp_liq
     .
     .
     .
end procedure
procedure mix_samp
     .
     .
etc.
```

2) Create a procedure for loading the file created in (1), above. Give it a name (for example, "gc_libr"). It will use the "library" command and be similar to the following:

```
procedure gc_libr
library fill_syr
end procedure
```

3) In your final PERL procedure, similar to "gc_prep", insert a line near the beginning which executes the procedure from (2), above, loading all the procedures needed:

```
gc_libr
```

SOME SAMPLE PROCEDURES

This section contains examples of procedures which perform some of the common procedures carried out by modules of the MasterLab System. Although you will have to modify them for your particular purposes, they may save you some programming time.

> **Important:** Except for a rare coincidence, the Directory in your system will not contain the user-created commands in these procedures. Be sure you use the Teach Programs to create such commands before you attempt to use them.

Getting Tubes from a Rack and Returning Them

The "Define a Rack" option of the robot Teach Program should be used to name and describe each rack in your system. The sample procedures which follow in this section will assume a system which contains a 40-tube rack called "sample", a 40-tube rack called "empties", and a 40-tube vial rack called "vial".

The format of the command to get a tube from a rack is

```
rackname i
```

where i is the number of the location of the tube in the rack. (See Section 3 on defining a rack.) To return the tube, the format is

```
rackname i back
```

Thus, to take the third tube from "empties" and then replace it, the commands would be

```
empties 3
empties 3 back
```

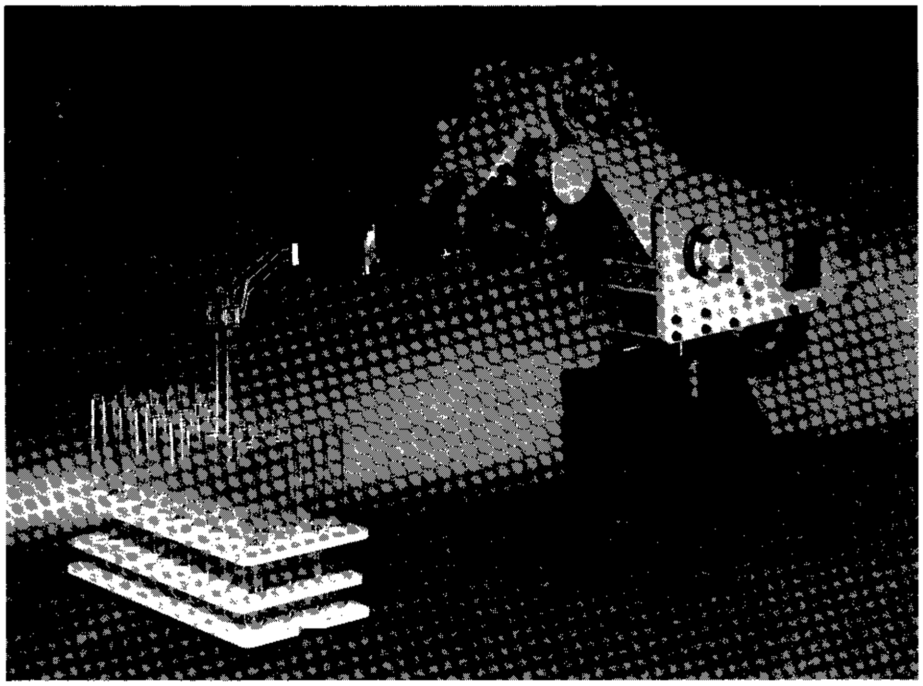


Figure 6-2 - Getting a Tube from a Rack

To position the robot hand above a location in the rack (for example, location 5), the command would be

```
empties 5 over
```

Using the MasterSyringe

Performing Dilutions

The following procedure draws in 1 ml of solvent and 0.5 ml of sample, separated by a 0.1 ml air bubble. (Refer to the discussion of the "fill_syr" procedure under "Planning New Procedures, or to the MasterSyringe manual, for an explanation of the dilution procedure.) The entire 1.6 ml could then be dispensed by the "dispense_liq" procedure, below.

```
procedure dil_samp
for i = 1 to 40
     sample i
     fill_1ml_solvnt
     fill_0.1_air
     under_disp
     rem  intermediate positions as tube is raised
     mid_dispenser_1
     mid_dispenser_2
     at_dispenser
     fill_0.5_sample
     rem  intermediate positions as tube is lowered
     mid_dispenser_2
     mid_dispenser_1
     under_disp
     sample i back
next i
end procedure
```

Dispensing Liquids

This procedure assumes that liquid has been drawn into the syringe (i.e. the plunger is not at zero). The robot gets a test tube, brings it to the dispensing stand, and raises it (in 3 steps) around the dispensing probe. The syringe plunger returns to zero, the test tube is lowered, and the tube is returned to its rack. Notice that the procedure must give the syringe about 5 seconds of time whenever it has to fill or empty.

```
procedure dispense
for i = 1 to 40
     empties i
     under_disp
     mid_dispenser_1
     mid_dispenser_2
     at_dispenser
     zero_plunger
     set timer 1 for 5 seconds
     wait for timer 1
     mid_dispenser_2
     mid_dispenser_1
     under_disp
     empties i back
next i
end procedure
```

Using a Syringe Variable

A syringe variable is a variable name which you store in the PERL directory by means of the syringe Teach Program. It becomes a reserved word, whose only permitted use in programs is to stand for a syringe plunger position parameter (i.e., a volume).

The procedure which follows assumes that you have named and stored a syringe variable called "syr_1_vol". The procedure fills syringe_1 with 5 ml of liquid, then dispenses 0.2 ml of that liquid to each of 25 test tubes. If the tube numbers (1 to 25) are represented by i, then you must define a variable ("a", for example) such that

```
a = 5.00 - 0.2*i
```

Then your syringe variable, syr_1_vol, would be equal to a. The PERL syntax for this is

```
syr_1_vol a
```

Those two lines would be used in a procedure similar to the one below. They cause the syringe plunger to be depressed 0.2 ml farther each time the loop in the procedure is executed, thus dispensing the reagent.

```
procedure use_var
valve_input
fill_syr_5ml
valve_output
for i = 1 to 25
     sample i
     under_disp
     mid_dispenser_1
     mid_dispenser_2
     at_dispenser
```

```
        a = 5.00 - 0.2*i
        syr_1_vol a
        set timer 1 for 5 seconds
        wait for timer 1
        mid_dispenser_2
        mid_dispenser_1
        under_disp
        sample i back
    next i
    end procedure
```

Note that you can define the value of syr_1_vol by other expressions in other procedures, but the system will always interpret it as referring to a syringe plunger position.

Mixing

Commands taught by the device interface Teach Program turn the mixer on and off. One of the PERL software timers (see Section 4B,2) times the duration of mixing.

The subroutine below also assumes that you used the "Define a Rack" option of the robot Teach Program to teach the robot that the mixer is a rack (named "mixer") having 1 column and 1 row. This allows you to use the rack commands "mixer 1" and "mixer 1 back".

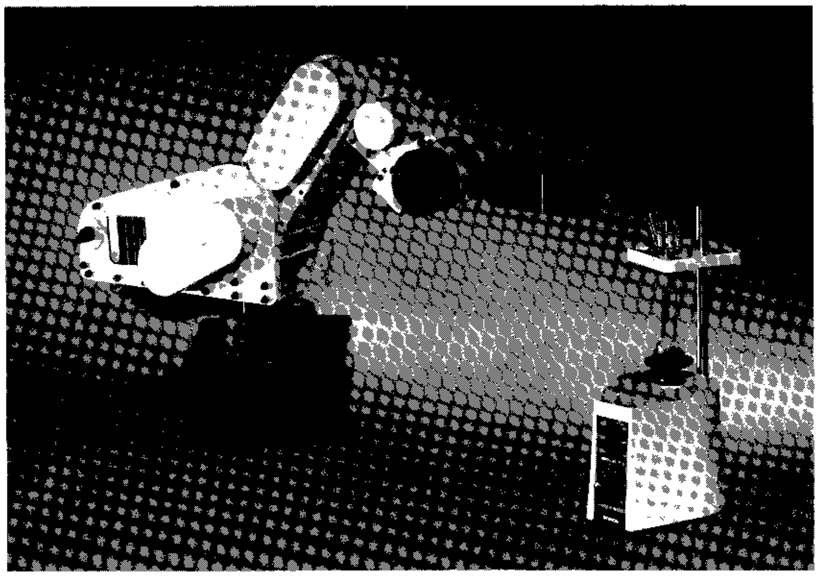


Figure 6-3 - Mixing

```
procedure mix_samp
mixer 1 back
turn_on_mixer
set timer 1 for 15 seconds
wait for timer 1
turn_off_mixer
mixer 1
end procedure
```

Crimping

When the robot is to use the crimper, it first gets a loosely capped vial from the vial rack and takes it to the vacuum device on the crimper. The vacuum removes the cap and retains it while the robot dispenses a sample into the vial. The robot then retrieves the cap from the vacuum and places the vial and the cap in the crimping jaws for crimping.

An input to the device interface from the crimper lets the device interface test whether the vacuum is on. If the vacuum does not turn on, you can program an alarm to sound.

In the following procedure, "pros_sam" stands for other procedures which would be carried out after the cap is left at the vacuum.

```
procedure crmp_vil
for i = 1 to 40
    vial i
    under_vac
    rem  raise vial to vacuum
    up_to_vac
    turn_on_vac
    rem  vacuum is a variable to test whether vacuum turns on
    until vacuum = 0
        vacuum = vacon
        if vacuum = 0 then
            sound alarm
        end if
    end until
    alarm_off
    pros_sam
    under_vac
    up_to_vac
    turn_off_vac
```

```
          until vacuum = 1
               vacuum = vacon
               if vacuum = 1 then
                    sound alarm
               end if
          end until
          alarm_off
          under_jaws
          up_to_jaws
          crimp
          under_jaws
          vial i back
     next i
     end procedure
```

Weighing

Balances have one-way communication; they output measured masses to the System Controller. The only kind of balance command which can be issued is a command to the System Controller to accept this reading.

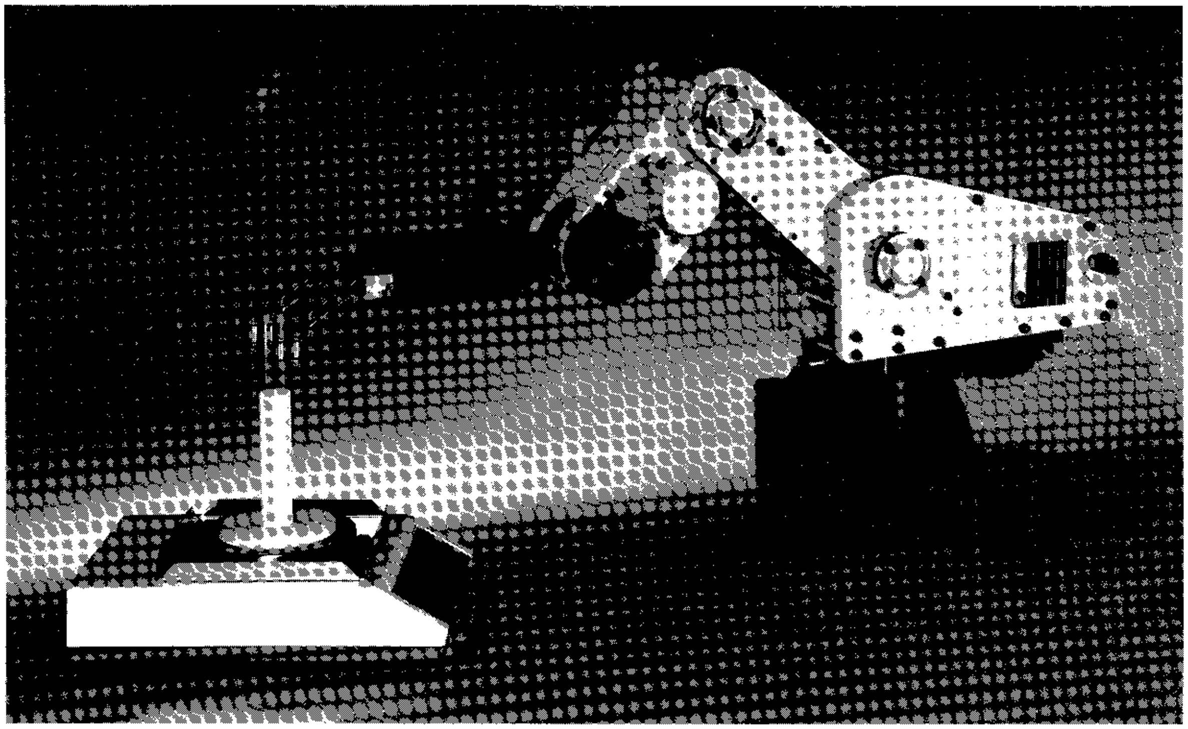


Figure 6-4 - Weighing

A balance command, such as "weigh_sample", causes the measured mass to be output to the System Controller. The following procedure weighs a sample, transfers the measured mass to a variable called "mass", then appends mass to a file on disk.

```
procedure stor_wt
mass = get_weight
append mass to mass_fil$
close mass_fil$
end procedure
```

APPENDICES

APPENDIX 1
ERROR MESSAGES

Invalid command
Command cannot be sent to the appropriate device
Invalid device name
Cannot resolve:
Insufficient memory for this program
Syntax error has occurred
Internal stack overflow has occurred
FOR stack overflow
WHILE stack overflow
UNTIL stack overflow
NEXT without a corresponding FOR
END WHILE without a WHILE
END UNTIL without an UNTIL
Invalid expression
Missing THEN
Missing END IF
ELSE without a corresponding IF
END IF without a corresponding IF
FOR without a corresponding NEXT
Missing END WHILE
Missing END UNTIL
Too many files open
Cannot open file
Cannot close file
File not found
Error occurred while reading from
Error occurred while writing to
Subscript out of range
Duplicate definition found
Mathematical error
A parameter quantity error has occurred

APPENDIX 2
USEFUL DOS COMMANDS

LEAVING PERL, ACCESSING DOS

When you wish to use DOS commands, you must leave the PERL environment. There are two ways to do this:

- To return to the DOS Operating System while keeping PERL in memory, type **dos** and press Enter. You can then issue DOS commands. When you are ready to return to PERL, type **perl** and press Enter. The PERL screen reappears without the modules having to be re-initialized.

  Note: Your system must have sufficient memory in order to be able to use the DOS mode within PERL.

- To abort PERL and return to DOS, type **system** and press Enter. You will then be able to issue DOS commands and to load other software.

DOS COMMANDS

The commands discussed below are DOS commands which perform essential housekeeping functions such as formatting and copying disks. If you will be doing any saving of programs and/or files on disk, you will need these commands. Be aware, however, that much more information is available to you, about these and many other DOS commands, in the Disk Operating System manual.

In the command formats which follow, parameters shown in square brackets are optional. The parameter symbolized by "d:" is the disk drive, and may be entered in the command line as either "A:" (the left drive) or "B:" (the right drive). You may have copied one or more of these commands onto your PERL/DOS work disk. The others are on your DOS disk.

<u>COPY</u> - Copies a file to another (or the same) diskette. The new file may be given a different name.

<u>Copy with the same filename</u>:

COPY [d:] filename [d:]

If the disk drives are not specified, the file will be copied to the current directory of the default drive. The source drive and the target drive must be different, since the new and old filenames are the same.

Example: COPY B:MYPROG A:

<u>Copy with a different filename</u>:

COPY [d:] filename [d:] filename

If no drives are specified, the default drive is used. The drives do not have to be different.

Example: COPY B:MYPROG B:NEWPROG

<u>Copy and combine</u>:

COPY [d:] filename + [d:] filename [d:] [filename]

If no target filename is given, all the specified files will be placed in the first file named. Otherwise, the files are concatenated and placed in the target file.

Example: COPY FILA + FILB + FILC BIGFILE

<u>DISKCOPY</u> - Copies the entire contents of the diskette in the source drive
onto the diskette in the target drive. The target disk will be formatted, if necessary.

DISKCOPY [d:] [d:] [/1]

The first parameter is the source drive, the second is the target. If the /1 parameter is used, only the first side of the diskette will be copied. If the two drives are the same, or if none are specified, a single-drive copy is performed; you will be prompted to insert appropriate diskettes as needed. After copying is complete, DISKCOPY asks

Copy another (Y/N)?

If you answer Y, the next copy will be performed using the same drives as were specified in the original command.

<u>ERASE</u> - Deletes a file from a diskette. DEL is an alternative form of the command.

```
ERASE [d:] filename
    or
DEL [d:] filename
```

<u>FORMAT</u> - Initializes the diskette in the designated drive or the default drive.

<u>Caution:</u> Formatting destroys all data on the diskette.

```
FORMAT [d:] [/S] [/1] [/8] [/V] [/B]
```

The optional parameters of the FORMAT command have the following meanings:

/S - causes the operating system files to be placed on the new disk.

/1 - causes the disk to be formatted only for single-sided use.

/8 - formats the diskette for 8 sectors per track. (The default is 9.)

/V - prompts you for a volume label which will be written on the disk to uniquely identify it.

<u>Note:</u> /8 and /V cannot both be entered at the same time.

/B - allows any system of DOS to be placed on the diskette. Otherwise, only versions 2.00 or 2.10 may be placed on it.

<u>Note:</u> /V and /S cannot be used with /B

Example: To format the disk in drive B for single-sided use, with a volume number, enter

```
FORMAT B:/1/V
```

The system displays the following message:

```
Insert new diskette for drive B:
and strike any key when ready
```

After you have carried out these instructions, the message

```
Formatting...
```

appears. When formatting is complete, the system prompts

```
Format complete

Volume label (11 character, ENTER for none)?
```

When you have responded to the prompt, the system will display

```
xxxxxx bytes total disk space
 xxxxx bytes used by system
xxxxxx bytes available on disk

Format another (Y/N)?
```

Enter Y to format another diskette, N to end the format program.

APPENDIX 3
CONTROL CHARACTERS AND THEIR PERL DEFINITIONS

Character Number dec	Character Number (hex)	ASCII Abbreviation	PERL Abbreviation	Alternate Abbreviation
Ø	Ø	NUL	<NUL>	
1	1	SOH	<SOH>	<CTRLA>
2	2	STX	<STX>	<CTRLB>
3	3	ETX	<ETX>	<CTRLC>
4	4	EOT	<EOT>	<CTRLD>
5	5	ENQ	<ENQ>	<CTRLE>
6	6	ACK	<ACK>	<CTRLF>
7	7	BEL	<BEL>	<CTRLG>
8	8	BS	<BS>	<CTRLH>
9	9	HT	<HT>	<CTRLI>
1Ø	A	LF	<LF>	<CTRLJ>
11	B	VT	<VT>	<CTRLK>
12	C	FF	<FF>	<CTRLL>
13	D	CR	<CR>	<CTRLM>
14	E	SO	<SO>	<CTRLN>
15	F	SI	<SI>	<CTRLO>
16	1Ø	DLE	<DLE>	<CTRLP>
17	11	DC1	<DC1>	<CTRLQ>
18	12	DC2	<DC2>	<CTRLR>
19	13	DC3	<DC3>	<CTRLS>
2Ø	14	DC4	<DC4>	<CTRLT>
21	15	NAK	<NAK>	<CTRLU>
22	16	SYN	<SYN>	<CTRLV>
23	17	ETB	<ETB>	<CTRLW>
24	18	CAN	<CAN>	<CTRLX>
25	19	EM	<EM>	<CTRLY>
26	1A	SUB	<SUB>	<CTRLZ>
27	1B	ESC	<ESC>	
28	1C	FS	<FS>	
29	1D	GS	<GS>	
3Ø	1E	RS	<RS>	
31	1F	US	<US>	
32	2Ø	SP	<SP>	<BLANK>
127	3F	DEL	<DEL>	

APPENDIX 4
SAMPLE PERL APPLICATION

MAIN PROCEDURE: DISKIO

```
rem
rem This example is of PERL's disk I/O capabilities as well as how to build a
rem turnkey PERL application.
rem
rem This procedure is the main procedure of our application.  It builds a menu
rem from which the user may select to write to disk or to read from disk what
rem was previously written.  There is a third option which is to EXIT from this
rem procedure and return to PERL.
rem

procedure diskio

rem
rem Set up the loop condition to loop through the WHILE loop until the EXIT
rem option is selected from the menu.  Once the EXIT option is selected,
rem EXIT_FLAG$ will have the value "y" and the WHILE loop will terminate.
rem

   exit_flag$ = "n"
   while exit_flag$ <> "y"
      clear

rem
rem After the screen is cleared, place the title of our procedure on the screen
rem centered with a box drawn around it.  It is important that the title be in
rem the variable title$ because the procedure box will use the value of title$
rem to center and draw the box for the header.  This is an example of how the
rem value of the variable exists beyond the current procedure that it is used
rem in.
rem

      title$ = "DISK I/O TEST"
      box

rem
rem Draw the menu
rem

      cursor 10,27
      display "1 - Read and display"
      cursor 11,27
      display "2 - Write data to datafile"
      cursor 13,27
      display "9 - Exit"
```

```
rem
rem Now that the menu is drawn, prompt the user for the menu item desired.  The
rem UNTIL loop is used to validate the input.  In other words, we only want the
rem user to enter a 1, 2 or 9 and everything else is invalid.  So if an invalid
rem item is entered, the user is re-prompted for the desired menu item.
rem

      until menu_item% = 1 or menu_item% = 2 or menu_item% = 9
         cursor 20,27
         input "Enter menu item desired : ",menu_item%
      end until

rem
rem A valid menu item has been entered, so now perform the appropriate process
rem
rem
rem Option 1 uses procedure FREADINP to read a disk file & display its contents
rem Option 2 uses procedure FWRITOUT to create a disk file
rem Option 9 will prompt the user if they are sure that they want to exit this
rem          procedure.  If they are then it will exit, otherwise the screen
rem          will be redrawn.
rem

      case menu_item%
      is 1 :
             freadinp
      is 2 :
             fwritout
      is 9 :
             resp = 0
             until resp = 1 or resp = 2
                cursor 23,1
                input "Are you sure (y/n)";resp$
                if resp$ = "y" then
                   resp = 1
                   exit_flag$ = "y"
                else
                   if resp$ = "n" then
                      resp = 2
                   end if
                end if
             end until
      end case
   end while
end procedure
```

SUBPROCEDURES

box

```
rem
rem This procedure displays the header for this application.  The header
rem consists of a centered box drawn around the title of the application.
rem The size of the box is directly dependent upon the size of the title.
rem This example shows the user characters that cannot be accessed from the
rem keyboard.  These characters are accessed by using their octal codes in
rem a string.  See your PERL user manual for further details.
rem

procedure box

rem
rem The variable title$ contains the title of this application.  It's value
rem is set by the invoking procedure.  We want to know the length of the
rem title in order to be able to center it.
rem

   title_size% = len(title$)
```

```
rem
rem We calculate the starting column for drawing the box.
rem

   start_col% = 38 - int((title_size% / 2) + .5)

rem
rem We build a string of "\315"'s. \315 is the octal code for a character that
rem is used as the straight line in our box.  It is actually 2 parallel lines.
rem

   strght_line$ = "\315\315\315\315\315\315\315\315\315\315\315\315\315\315"

rem
rem This line allows us to increase the length of the string by using string
rem concatenation operations.
rem

   strght_line$ = strght_line$+strght_line$+strght_line$+strght_line$
rem
rem Draw the top of the box. \311 is the upper left corner character and \273
rem is the upper right corner character.
rem

   cursor 1,start_col%
   display "\311" + left$(strght_line$,title_size%+2) + "\273"

rem
rem Draw the title line in the box, including the box lines (\272).
rem

   cursor 2,start_col%
   display "\272 " + title$ + " \272"

rem
rem Draw the bottom of the box. \310 is the lower left corner character and
rem \274 is the lower right corner character.
rem

   cursor 3,start_col%
   display "\310" + left$(strght_line$,title_size%+2) + "\274"

rem
rem Now that the box is drawn, the procedure performs an implicit return to
rem the invoking procedure.
rem

end procedure
```

freadinp

```
rem
rem This is procedure freadinp. It reads the contents of a disk file and disp-
rem lays it to the console.  The name of the disk file is TESTFILE and it is
rem hardcoded in our example.  It should be noted that the file name may be
rem replaced by a string variable whose value is the file name.  This is left
rem as an exercise for the reader.
rem

procedure freadinp
   clear

rem
rem Set the loop condition so that the procedure will loop in the WHILE loop
rem until a null("") is read from the file. When a null is read, then exit
rem from the loop.
rem

   ans$ = " "
   while ans$ <> ""
      read ans$ from "testfile"
      display ans$
   end while
```

```
rem
rem A null has been read so close the file.
rem

   close "testfile"

rem
rem Now that the file is closed, allow the user to view the screen for as
rem long as they like.  This is accomplished by prompting the user for
rem input.  The input, in this case any key, signifies the user's desire to
rem continue with the procedure.  In our example, the procedure performs an
rem implicit return to the procedure that invoked it.
rem

   display
   display "Press a key to continue..."
   a$ = inkey$
end procedure
```

fwritout

```
rem
rem This procedure prompts the user for string data and then places it on disk.
rem The disk file name is hardcoded and it can be replaced with a string var-
rem iable whose value is the file name.
rem

procedure fwritout
   clear

rem
rem Set the condition for looping through the WHILE loop until a null("") is
rem entered by the user. A null is entered by pressing the RETURN key at the
rem prompt.  The null is stored in the disk file.
rem

   ans$ = " "
   while ans$ <> ""
      input "Enter string to place on disk : ",ans$
      write ans$ to "testfile"
   end while

rem
rem A null was entered so close the data file and end the procedure.
rem

   close "testfile"

rem
rem Once the file is closed, the procedure performs an implicit return to the
rem invoking procedure.
rem

end procedure
```

APPENDIX 5
PERL RESERVED WORDS

You can use the words listed below as part of command, variable, or procedure names, but not at the beginning of the name. For example, the name "breakkey" is illegal, but "keybreak" is legal.

break	case	close	date$	day$
devices	display	dos	else	if
inkey$	month$	open	parallel	perl
print	procedure	readkey$	rem	save
screen	system	then	time$	timer
until	while			

The following words can form part of a variable, procedure, or command name, but do not use them as names exactly as written. For example, "append" is not a legal name, but "append1" would be legal.

append	as	close	cursor	define
eof	for	from	init	input
library	log	next	output	read
receive	redirect	rewind	send	set
speed	step	to	use	wait
write				

The characters that follow have specific functions in PERL. Do not use them in a command, variable, or procedure name unless you intend them to perform their functions.

! - + * / $ %^ & ( ) \ < > " ? ' @ = |

PERL STATEMENTS

The number at the end of each definition is the page where you will find detailed instructions for using the statement.

Function	Definition
`append (var) to (filename)`	Adds the variable to the end of the previously-existing sequential file named. The filename argument may be an expression. (4-40)
`as`	See "define". (4-39)
`break`	Transfers control out of a loop. Control goes to the next statement immediately following the loop in which the "break" occurs. (4-37)
`case (expr) is(const): (statement)... default: (statement) end case`	Specifies which of several actions will be taken, depending on the value of the expression. (4-32)
`clear`	Erases the monitor screen. (4-24)
`close`	Closes the robot grippers. (4-18)
`close (filename)`	Closes the sequential file specified. The filename argument may be an expression. (4-40)
`continuous (robot positions) end continuous`	Moves the robot through the specified positions without stopping. (4-19)
`cursor (h line no.), (v column no.)`	Moves the cursor to the specified location on the monitor screen. Line and column numbers may be expressions. (4-24)
`date$`	Outputs the date to the screen, or appends it to a file. (4-43)
`day$`	Outputs the day of the week to the screen, or appends it to a file. (4-43)
`default`	See "case". (4-32)
`define (var) as (port name)`	Establishes the buffer var to contain the serial input/output of the named port. (4-39)
`devices`	See "display devices". (4-17)
`dim (var(i))`	Allocates memory for the array var(i), and specifies that i is its largest subscript. A variable may have up to three dimensions. (4-36)
`display (desired output)`	Specifies and formats output to the monitor. (4-24)
`display devices`	Displays the configurations for all devices in the system configuration file. (4-17)
`dos [command or batchfile]`	Returns control to the operating system. PERL remains in memory. (4-13)
`down (mm)`	Moves the robot down a specified distance from its present position. (4-18)
`else`	See "if". (4-31)
`end procedure`	Statement which ends every procedure and subroutine. (4-11)
`eof`	End-of-file function. Checks to see if the present record is the last in the file. (4-42)
`for (var) = (expression) to (expression) [step (expression)]`	First line of a program loop. (4-34)
`from`	See "redirect" (4-28) and "read" (4-40).
`if (expression) then (statement) [else (statement)] end if`	Allows different actions to follow, depending on whether the expression is true or not. (4-31)

Reference guide to PERL

The following alphabetical listing gives a format and a brief summary of the use of each PERL function or statement. For complete descriptions, with examples, see the indicated pages in Section 4 of this manual.

NUMERIC FUNCTIONS

In using these functions, express all angles in radians. See page 4-5 for examples.

Function	Definition
`abs(x)`	Absolute value of x.
`arccos(x)`	Arccosine, in radians, of x.
`arcsin(x)`	Arcsine, in radians, of x.
`arctan(x)`	Arctangent, in radians, of x.
`cos(x)`	Cosine of x radians.
`csc(x)`	Cosecant of x radians.
`cot(x)`	Cotangent of x radians.
`exp(x)`	Exponent; returns e to the xth power.
`exp10(x)`	Exponent to base 10; returns 10 to the xth power.
`int(x)`	Integer part of x. x will be truncated; to round, use int(x + 0.5).
`log(x)`	Natural logarithm of x.
`log10(x)`	Logarithm to base 10 of x.
`sin(x)`	Sine of x radians.
`sqrt(x)`	Square root of x.
`tan(x)`	Tangent of x radians.

STRING FUNCTIONS

For examples of these functions, see pages 4-6 and 4-7 of this manual.

Function	Definition
`+`	Concatenate strings.
`asc(x$)`	ASCII code for first character of x$.
`chr$(x)`	Character with ASCII code x.
`x$ = inkey$`	Pauses until a character is input from the keyboard, then assigns it to x$.
`instr(x$,y$)` `instr(n,x$,y$)x$.`	The position of the 1st occurrence of y$ in x$. May begin searching at the nth position.
`left$(x$,n)`	The left n characters in x$.
`len(x$)`	Length of string x$.
`mid$(x$,i)` `mid$(x$,i,n)`	n characters beginning at position i in x$.
`x$ = readkey$`	Accepts a character input from the keyboard, but does not pause if none is input.
`right$(x$,n)`	The rightmost n characters in x$.
`str$(x)`	String constrant represent-ing the numeric value x.
`val(x$)`	Numeric value of string x$.

PERL STATEMENTS

The number at the end of each definition is the page where you will find detailed instructions for using the statement.

Function	Definition
`init (device)`	Causes initialization of the device named. (4-17)
`inkey$`	Pauses to accept a character input from the keyboard. (4-28)
`input [prompt] (variable)`	Pauses until a variable is input from the keyboard. (4-25)
`is`	See "case". (4-32)
`library`	Loads a previously-created file containing all the subprocedures used in a main procedure. (4-15)
`link (procedure name)`	Links all the subprocedures used in a main procedure. (2-18)
`load`	Reads the PERL Directory from disk, and makes a backup copy in the file PERL.BAK. (4-14)
`log (destination)`	Creates a file on disk which will contain everything sent to the screen. 4-16)
`month$`	Outputs the present month to the screen, or appends it to a file. (4-43)
`next (var)`	Last line of a program loop which begins with "for". (4-34)
`open`	Opens the robot grippers. (4-18)
`output`	See "redirect". (4-28)
`perl`	Returns control to the executing procedure after use of the "dos" command. (4-13)
`print`	Sends subsequent output to the printer. (4-30)
`print screen`	Outputs the entire present contents of the screen to the printer. (4-30)
`procedure (name)`	Begins every procedure and subroutine. (4-11)
`read (var) from (filename)`	Read the next record from the file into memory. The filename argument may be an expression. (4-40)
`readkey$`	Reads a character from the keyboard buffer if a key has been pressed. (4-29)
`receive (var)`	Reads from a defined serial port into the buffer named var. (4-39)
`redirect input from (device)` <u>or</u> `to (device)`	Changes the standard input or output device until the next redirect statement. (4-28)
`relative (x,y,z,p,r)`	Moves the robot relative to its present position. The arguments specify m in the x, y, and z directions and degrees of pitch and roll. (4-18)
`rem (comment)`	Begins a comment. Use "rem" at the beginning of a line, or " ! " to append a comment to the end of a line. (4-11)
`rewind (filename)`	After reading or writing to a file, return to the beginning. (4-40)
`save`	Writes the Directory in memory to the disk file PERL.DIR. (4-14)
`screen`	See "print screen" (4-30)
`send (var)`	Writes the contents of buffer var to a defined serial port. (4-39)
`set timer (no.) for (quantity of time) (type of time)`	Sets one of the 10 software timers. (4-20)

PERL STATEMENTS

The number at the end of each definition is the page where you will find detailed instructions for using the statement.

Function	Definition
`speed (integer or variable)`	Sets the speed of the robot; 0 is slowest, 9 is fastest. (4-19)
`step`	See "for". (4-34)
`stop`	Aborts procedure execution and returns to the DCP. (4-14)
`suspend`	Halts procedure execution until you either continue (F8) or abort (F7). (4-14)
`system`	Returns control to the operating system; aborts PERL. (4-14)
`then`	See "if". (4-31)
`time$`	Outputs the present time of day to the screen, or appends it to a file. (4-43)
`timer`	See "set timer". (4-20)
`to`	See "for" (4-34)
`until (condition) (statements) end until`	Carry out the statements until the condition is true. (4-22)

Function	Definition
`up (distance)`	Move the robot up from its present position a specified distance. (4-18)
`use (device name)`	Tells which of two similar modules (e.g. robots) is to be used. (4-17)
`wait for timer (number)`	Wait for the time set on the specified timer to elapse. (4-20)
`while (condition) (statements) end while`	Carry out the statements as long as the condition is true. (4-22)
`write (var) to (filename)`	Places the variable in the new sequential file named. The filename argument may be an expression. (4-40)
`!`	Signals a comment appended to a line. See "rem". (4-12)
`\(code for character)`	Accesses the IBM PC character set. (4-26)

RELATIONAL AND LOGICAL OPERATORS

Operator	Operation
=	equal to
<>	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
AND	logical AND
OR	logical OR
NOT	logical NOT

ARITHMETIC OPERATORS

Operation	Operator	Expression
Addition	+	a + b
Subtraction	-	a - b
Multiplication	*	a*b
Division	/	a/b
Modulus	@	a @ b
Negation	-	-a